**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**
**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**
**імені ІГОРЯ СІКОРСЬКОГО»**
**ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ**
Кафедра інформаційної безпеки

«На правах рукопису»

УДК 519.865.3

«До захисту допущено»
В.о. завідувача кафедри
_____ М.В.Грайворонський
“___”_____2019 р.

# Магістерська дисертація

**на здобуття ступеня магістра**

зі спеціальності: 113 Прикладна математика

на тему: «Рефлексивна модель асиметричної гри полковника Блотто»_____
_____

Виконав: студент 2 курсу, групи ФІ-72 мп
(шифр групи)

Красношлик Костянтин Юрійович _____ _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник Смирнов Сергій Анатолійович_____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _____ _____ _____
(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

Рецензент_____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019 року

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**
**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**
**імені ІГОРЯ СІКОРСЬКОГО»**
**ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ**
Кафедра інформаційної безпеки

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою
Спеціальність (спеціалізація) – 113 Прикладна математика («Аналітичні методи безпеки інформації»)

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ М.В.Грайворонський
(підпис)

«___»_____2019 р.

**ЗАВДАННЯ**
**на магістерську дисертацію студенту**
Красношлику Костянтину Юрійовичу
(прізвище, ім'я, по батькові)

1. Тема дисертації «Рефлексивна модель асиметричної гри полковника Блотто»

науковий керівник дисертації - к.ф.-м.н. Смирнов Сергій Анатолійович,
_____ _____ ,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «15 » листопада 2018 р. № 4171-с

2. Термін подання студентом дисертації 06.05.2019

3. Об'єкт дослідження _____
_____

4. Предмет дослідження _____
_____
_____
_____

5. Перелік завдань, які потрібно розробити _____
_____
_____
_____
_____
_____

6. Орієнтовний перелік ілюстративного матеріалу _____
_____
_____

7. Орієнтовний перелік публікацій _____
_____

8. Консультанти розділів дисертації*

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| | | | |
| | | | |
| | | | |

9. Дата видачі завдання _____

Календарний план

| № з/п | Назва етапів виконання магістерської дисертації | Термін виконання етапів магістерської дисертації | Примітка |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

Студент _____ (підпис) _____ (ініціали, прізвище)

Науковий керівник дисертації _____ (підпис) _____ (ініціали, прізвище)

* Консультантом не може бути зазначено наукового керівника магістерської дисертації.

# РЕФЕРАТ

Обсяг роботи: 95 сторінок, 2 ілюстрації, 6 таблиць, 1 додаток, 19 джерел за переліком посилань.

Метою роботи є аналіз переваг та недоліків існуючих моделей гри полковника Блотто, та розробка методу, що знаходить множину оптимальних рішень розподілу ресурсів для цієї гри.

Об’єктом дослідження є множина стратегій розподілу ресурсів між декількома об’єктами у ГПБ, в якій гравці можуть мати різну кількість ресурсів та власні бачення важливості перемог на об’єктах.

Предметом дослідження є модель гри полковника Блотто та метод, що шукає множину близьких до оптимальних стратегій розподілу ресурсів у грі.

В процесі виконання роботи були розглянуті існуючі моделі ГПБ та дослідження присвячені ним. Був проведений аналіз переваг та недоліків існуючих методів пошуку оптимальних рішень гри та на основі цього було запропоновано свій метод, який додатково враховує випадки, де гравці мають різну кількість ресурсів та власні корисності від заволодіння об’єктами. Проведено дослідження нового методу та реалізовано програмний додаток. Результати роботи вказують на ефективність використання даного методу для пошуку оптимальних стратегій у ГПБ.

Одержані результати можуть бути використані під час моделювання ситуацій, які можливо описати в межах гри Блотто, та шляхом застосування запропонованого методу отримати множину оптимальних рішень гри. А це в свою чергу може збільшить шанси на отримання кращого виграшу.

Матрична гра, гра полковника Блотто, стратегія, розбиття чисел, аукціонна модель, рівновага Неша, розташування, об’єкти, вектор розподілу ресурсів.

# ABSTRACT

The volume of work is 95 pages. Contains: 2 drawings, 6 tables, 1 application and 19 references.

The aim of the work is to analyze the advantages and disadvantages of existing models of the Colonel Blotto Game, and to develop a method that finds the set of optimal solutions for resource allocation in this game.

The object of the study is a plurality of strategies for allocating resources between several objects in the GPB, in which players may have different amounts of resources and their own vision of the importance of winning objects.

The subject of the study is the model of the Colonel Blotto Game and a method that searches set of close to optimal resource allocation strategies in the game.

In this work were observed some of existing GPB models and research that were devoted to them. An analysis was made of the advantages and disadvantages of existing methods for finding optimal solutions of game, and on the basis of this, a new method was proposed which additionally takes into account cases where players may have different amounts of resources and their own utilities from ownership of objects in game. A new method was investigated and a software application was implemented. The results indicate the effectiveness of using this method to find optimal strategies in the GPB.

The obtained results may be used to simulate situations that can be described within the Blotto game, and to obtain a plurality of optimal game solutions by applying the proposed method. And this may increase the chances of getting a better winnings.

Matrix game, Colonel Blotto Game, strategy, number split, auction model, Nash equilibrium, location, objects, vector with resources distribution.

# ЗМІСТ

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ГПБ – гра полковника Блотто.

ДП – динамічне програмування.

ЗПР – задача пакування рюкзака.

NP – клас складності, до якого належать задачі, для яких завжди існує можливість їх успішного обчислення з використанням недетермінованих алгоритмів за поліноміальний час відносно довжини вхідного рядка.

# ВСТУП

За останні роки можна спостерігати суттєвий зріст популярності теорії ігор. Загальновідомо, що теорія ігор дає прекрасні результати в прикладній математиці, вона застосовується для вивчення поведінки людини і тварин в різних ситуаціях. Спочатку методи теорії ігор розвивалася в рамках економічної науки, вивчаючи поведінку економічних агентів в різних ситуаціях. Трохи пізніше теорія ігор почала знаходити своє місце в інших загальних науках, таких як, соціологія, політологія, психологія, етика та інших, де за допомогою якої деколи можна отримати пояснення поведінки людей.

В економіці вона може бути застосована не тільки для вирішення загальногосподарських завдань, а й для аналізу стратегічних проблем галузей, ринків, підприємств, розробок організаційних структур, систем управлінського обліку, ефективного розподілу ресурсів. За допомогою теорії ігор надається можливість змоделювати та передбачити дії своїх партнерів і конкурентів.

Ринкова економіка вимагає ефективного використання ресурсів суб'єктів. Оптимальний розподіл ресурсів є основною та обов'язковою умовою для стабілізації економіки. Цей факт привів до підвищення інтересу до завдань оцінки і пошуку оптимального розподілу ресурсів.

У той же час дії суб'єктів ринку завжди пов'язані з ризиком і конфліктом інтересів. Тому завдання оптимізації розподілу ресурсів часто розглядають з теоретико-ігрових позицій. Розподіл і використання обмежених ресурсів часто зустрічає організовану протидію конкурента, переслідує протилежні цілі.

Однією з моделей конфлікту, що має велику практичну значимість завдяки простоті побудови і можливостям опису широкого кола ситуацій, є модель матричних ігор. Дослідження можливості теоретико-ігрової

оптимізації розподілу ресурсів на основі апарату матричних ігор і їх змішаного розширення є актуальним завданням.

**Актуальність роботи.** Великий інтерес зараз полягає у вивченні поведінки людей у відносно складних стратегічних ситуаціях. Більшість експериментальних досліджень із стратегічного мислення зосереджено на поведінці в простих іграх, в яких набір стратегій або малий, або природно впорядкований. Ці ігри часто викликають привабливі правила прийняття рішень, такі як гра з рівновагою Неша (у чистих стратегіях), усунення ітеративно домінуючих стратегій або додавання додаткових умов для стратегій. Є певна потреба знайти схеми стратегічного мислення, що можуть бути використані в ситуаціях, коли важко оцінити простір стратегій через його розміри і структури. У таких ситуаціях, коли немає простого ментального уявлення про простір стратегій, гравець змушений формулювати свою стратегію, відштовхуючись від кількох особливостей таких стратегій. Крім того, виграш гравця залежить нетривіальним чином від функцій, що були обрані іншим гравцем. Теорія вибору в таких ситуаціях може надати інструменти для побудови нових моделей поведінки в різних контекстах.

У цій роботі було проведено дослідження декількох варіантів відомої гри полковника Блотто, що як раз розглядає задачу ефективного розподілу ресурсів між певною кількістю об’єктів двома гравцями. Гра сама по собі цікава і застосовується для вивчення різних економічних і політичних ситуацій. Гра також нагадує інші стратегічні сценарії, такі як розподіл коштів між різними завданнями, різноманітні характеристики продуктів та багатоцільові аукціони.

**Мета і завдання дослідження.** Основною метою цієї роботи є аналіз переваг та недоліків існуючих моделей гри полковника Блотто, та розробка методу, який би пропонував гравцеві варіанти оптимальних стратегій розподілу ресурсів, за умов що кожен із двох гравців може мати власні

вектори корисності об’єктів, що розглядаються у грі. Після того як цей алгоритм буде побудовано, його потрібно дослідити.

***Об’єкт дослідження:*** стратегії розподілу ресурсів між декількома об’єктами у грі, де за отримання переваги на цих об’єктах може змагатися дві особи, що можуть мати різну кількість ресурсів та власні вектори корисності таких об’єктів.

***Предмет дослідження:*** модель гри полковника Блотто та метод, за допомогою якого можна отримати одну або декілька стратегій, що є близькими до оптимального розташування ресурсів у грі.

**Методом дослідження** розділи з теорії ігор, комп’ютерні методи обчислення, рефлексивний аналіз.

**Наукова новизна одержаних результатів.** Запропоновано та досліджено новий метод пошуку множини близьких до оптимальних стратегій розташування ресурсів в умовах не рівних кількостей ресурсів у гравців та наявності власних векторів корисності об’єктів для них. Серед існуючих джерел відкритої інформації подібних моделей знайдено не було.

**Практичне значення одержаних результатів.** Можливість застосувати надану модель для отримання векторів розподілу ресурсів, що є близькими до оптимальних. Використання таких векторів у грі полковника Блотто може надати перевагу на об’єктах та збільшить шанси отримання кращого виграшу. Напрямками застосування можуть бути не тільки військові, але й економічні, політичні сфери, та будь-які інші де має місце конкурентний розподіл ресурсів.

# 1 ТЕОРЕТИЧНІ ВІДОМОСТІ ЗА НАПРЯМКОМ ДОСЛІДЖЕННЯ

Перед початком виконання даної роботи для її якісного розуміння є доречним повторити та узагальнити деякий теоретичний матеріал, без розуміння якого важко зрозуміти деякі моделі, об’єкти та кроки, що будуть описані у наступник розділах. Головним із загального теоретичного матеріалу є основи теорії ігор.

## 1.1 Основні поняття із теорії ігор

**Некооперативна гра** — це такий тип гри у теорії ігор, де гравці приймають рішення, що є незалежними один від одного. Точніше, некооперативна гра – це така математична модель взаємодії певного числа сторін (гравців), в процесі якої між гравцями не має можливості сформувати коаліції та об’єднати дії різних гравців одним спільним планом.

**Стратегія** гравця в грі - це план дій/реакцій на всі можливі події, що можуть відбуватися у грі, серед яких також можуть бути дії інших гравців. Маючи певну стратегію гравець завжди на протязі гри готовий зробити свій крок у відповідь на будь-яку обставину, що може мати місце відбутися на протязі гри. [1]

Набір стратегій – це множина певної кількості стратегії для усіх гравців, що може цілком повно описати усі можливі події на протязі гри. Набір стратегій повинен мати в собі тільки одну стратегію для кожного з гравців.

**Чиста стратегія** надає повну інформацію стосовно того, яким чином гравець буде реагувати на події у грі. Маючи таку стратегію людина буде знати які наслідки можуть виплисти у результаті певного вибору гравцем. Множиною усіх чистих стратегій, що можуть бути вибрані гравцем, називається **простором стратегій**.

**Змішана стратегія** вказує на ймовірність вибору кожної з чистих стратегії гравця. Інакше кажучи, гравець обирає одну з чистих стратегій із ймовірністю що зазначена у змішаній стратегії. Значення ймовірностей є незмінними та визначаються на усю гру перед її початком. Виходячи з цього можна зробити висновок що чисті стратегії є частковим випадком змішаних стратегій. В них ймовірність вибору однієї зі стратегій дорівнює одиниці, в той час коли ймовірність вибору інших - нуль, тобто одна єдина стратегія буде обиратися завжди на протязі гри.

**Функція виграшу** певного суб'єкта – це така функція, що може залежати від багатьох параметрів системи та самого суб'єкта, його стану, прийнятих рішень. Її результатом є очікуваний виграш гравця, при певних умовах що розглядалися, та входили у функцію в якості її параметрів.

Якщо порівняти суму виграшів усіх гравців, та суму їх програшів у грі, то за результатом цього порівняння можна зробити наступну класифікацію ігор:

- ігри з нульовою сумою, така ситуація трапляється при умові коли різниця між двома сумами дорівнює нулю. Для випадку з двома гравцями це було б наступним чином: виграш одного гравця дорівнює програшу іншого. Це наштовхує на думку про пряму конкуренцію між гравцями;
- ігри з постійною різницею. Для цих ігор можливі випадки, коли усі гравці можуть одночасно виграти, або програти. Через що іноді є сенс для гравців діяти спільно, для покращення результатів виграшу;
- ігри з ненульовою сумою. Цей тип є певним об'єднанням першого та другого типу. В таких іграх між гравцями можливі конфлікти або співпраця.

**Антагоністичною** грою, інакше кажучи грою з нульовою сумою, називають безкоаліційну гру, в якій приймають участь двоє гравців. Виграш одного з цих гравців дорівнює програшу іншого, тобто вони є протилежними.

Якщо описувати антагоністичну гру використовуючи математичний підхід, то ця гру можна буде описати використовуючи наступну трійку математичних об'єктів $<X,Y,F>$, де в якості $X$ і $Y$ виступають множини

стратегій для першого та другого гравців відповідно; а $F$ – є **функцією виграшу** для першого учасника. Аргументами функції $F$ є двійки стратегій (x, y), де x належить до множини X, а y - до Y. Результатом $F$ є дійсне число, що співвідносить вигоди першого гравця при реалізації стратегії x першим гравцем, а y - другим. Через те що гра є з нульовою сумою, то і взаємозалежність між функціями виграшів двох гравців будуть виражатися наступним чином $F_1 = - F_2$. Це демонструє протилежні інтереси гравців, те що в цій грі програш одного гравця дорівнює виграшу іншого.

Якщо повернутися до історичних фактів, то цей клас ігор став одним із перших що був розглянутим у теорії ігор, ще й з використанням різноманітних математичних моделей. Прикладом з життя, що влучно підходило б до класу антагоністичних ігор були азартні ігри. Можливо через вибір такого предмету дослідження цей розділ математики і отримав свою назву – теорія ігор.

Ігри поділяються на матричні, біматричні, безперервні та ін.

**Матричною грою** називається кінцева гра двох гравців з нульовою сумою, причому кожен з гравців має кінцеве число стратегій. Для зручності позначимо одного з гравців через $A$, іншого - через $B$. Нехай у гравця $A$ є m стратегій – $A_1,..,A_m$, а у гравця $B$ - n стратегій - $B_1,..,B_n$. Нехай гравець $B$ вибрав стратегію $B_k$, а вибір гравця $A$ пав на вибір стратегії $A_i$. Будемо вважати, що вибір учасниками гри стратегій $A_i$ і $B_k$ однозначно визначає результат гри – виграш $b_{i\ k}$ гравця $B$ і виграш $a_{ik}$ гравця $A$, за умови, що ці виграші пов'язані рівністю: $a_{ik} = - b_{ik}$ [2].

Негативний виграш називають зазвичай програшем. У даній ситуації видно, що виграш одного учасника дорівнює виграшу іншого учасника, взятого зі знаком мінус. Це дозволяє нам розглядати виграші тільки одного з гравців. Для прикладу візьмемо виграші гравця $A$.

Результати виграшу $a_{ik}$ в кожній можливій ситуації { $A_i$, $B_k$ }, де $i=1,2,...,m$, $k=1,2, ...,n$ можна записувати у вигляді матриці:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}. \quad (1.1)$$

Матриця А з розмірністю *m*n* і буде називатися **матрицею гри**, або платіжною матрицею (звідки і пішла назва гри - матрична). Слід зауважити, що в матричних іграх інтереси гравців прямо протилежні, що є ознакою антагоністичних ігор.

Завдання теорії ігор - визначити вибір стратегій двох гравців, при яких першому гарантується максимальний середній виграш, а другого - програш.

**Біматричні ігри** - це кінцеві ігри з ненульовою сумою. Де кожен гравець має кінцеве число стратегій. Так, наприклад, перший гравець може вибрати будь-яку з m своїх стратегій, які позначимо номерами $i = 1, 2, ..., m$, а другий - одну з n своїх стратегій, які ми позначимо через номера $j = 1, 2, ..., n$. Тоді якщо I учасник вибрав свою *i*-у стратегію, а II учасник - свою *j*-у, то в підсумку перший матиме виграш $a_{ij}$, а другий - $b_{ij}$ (умова, що $a_{ij} = -b_{ij}$, є необов'язковою).

Отже, кінцева гра з ненульовою сумою буде представлена у вигляді двох рівних платіжних матриць (за рахунок чого і отримала свою назву):

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \text{ и } B = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \dots & \dots & \dots & \dots \\ b_{m1} & b_{m2} & \dots & b_{mn} \end{pmatrix}. \quad (1.2)$$

**Рівновагою Неша** у грі, де кількість гравців більше або дорівнює двом, називається набір стратегій, використовуючи які усі з гравців реалізовують оптимальну стратегію, щодо дій інших учасників (їх стратегій). Водночас притримуючись цієї сукупності стратегій та виграшів що вони дають – не один з гравців не може покращити свій виграш, через зміну своєї першої стратегії, за умов коли жоден з інших гравців свій вибір не змінює. Термін рівноваги Неша був отриманий через людину що першою розглянула таку ситуацію в іграх. Ім’я цієї людини - Джон Неш [3].

Припустимо, що ми маємо наступну пару {*S, f*}, яка описує гру *n* учасників у нормальній формі, тут виступають $S_i$ в якості набору стратегій і-го гравця, звідки виходить що $S = S_1 \times S_2 \times .. \times S_n$ - множина всіх можливих чистих стратегій. З приводу *f* то вона дорівнює $(f_1(x),..,f_n(x))$ — як сукупність виграшів, при умові що x приймає значення на множині S. Якщо кожен учасник гри *i,* значення якого лежать на множині від одного до *n* вибере стратегію $x_i$ в профілі стратегій $x=(x_1,..,x_n)$, то учасник гри *i* отримає виграш $f_i(x)$. Важливим ще є той факт, що виграш одного гравця залежить від повного профілю стратегій: а не тільки від стратегії, вибраної самим гравцем *i*, але й від стратегій інших гравці, що приймають участь у грі. Профіль стратегій x* що належить множині S можна вважати рівновагою по Нешу, в тому випадку коли зміна гравцем своєї першочергової стратегії з $x^*$ на $x_i$ є не вигідною для усіх з гравців, тобто:

$$\forall i,\ x_i \in S_i : f_i(x_i^*, x_{-i}^*) \geq f_i(x_i, x_{-i}^*). \qquad (1.3)$$

Існує два різновиди рівноваги Неша в залежності від знаку, що стоїть у нерівності вище: сувора рівновага Неша та слабка. Якщо в нерівності стоїть знак строго більше (>) при порівнянні з усіма парами гравців, та різновидами їх можливих альтернативних стратегій, то ця рівновага вважається суворою. А при умові що хоча б в одному з порівнянь утворюється рівність між $x_i^*$ та деякою іншою стратегією з множини *S*, то цю рівновагу класифікують як слабка рівновага Неша.

У грі іноді є можливість знайти рівновагу Неша в чистих або в змішаних стратегіях. З приводу останнього випадку, чиста стратегія обирається ймовірнісним чином із наперед заданою ймовірністю).

Ще одним з доведених фактів від Джона Неша є те, що якщо дозволити змішані стратегії, тоді в будь-якій грі з певною скінченної кількістю гравців, що обирають свої стратегії серед значень скінченної множини стратегій, то серед них хоча б одна буде рівновагою Неша.

## 1.2 Дилема в'язня

Одним із найвідоміших прикладів з теорії ігор є дилема в’язня. По собі вона є грою з ненульовою сумою, у центрі подій якої є гравці що заради отримання виграшу можуть співпрацювати один з одним або навпаки зраджувати. Тут, як і у інших прикладах теорії ігор, вважається, що учасники гри («в'язень») намагаються максимізувати свої власні виграші, не турбуючись за виграші інших.

Можливо на відміну від інших прикладів, у дилемі в’язня з більшою ймовірністю буде вибрана стратегія зрадити над тією стратегією, де гравці йдуть на співпрацю. Звідси випливає що єдиною рівновагою системи є випадок, при якому обоє учасників зраджують один одного. Кажучи іншими словами – не має великого значення яке рішення прийме другий учасник, через те, що для кожного з них є вигіднішим зрадити опонента. Та виходячи з цього, та й прийняття тої гіпотези, що учасники є раціональними гравцями, то незалежно від умов у ситуації гравцями буде обрана стратегія зради інших гравців [4].

Приймаючи рішення відштовхуючись від своїх власних інтересів, тобто обираючи рішення, яке є раціональним тільки для себе, гравці в цілому отримають неоптимальне рішення для них обох одночасно. Тобто скоріш за все, їх вибір зупиниться на зраді опонента, а в цьому випадку виграш становитиме менше значення за той, коли усі гравці співпрацюють (єдина рівновага в цій грі не веде до Парето-оптимального рішення). Ця різниця й утворює дилему в’язня.

Для випадків періодичного повторення дилеми в’язня, додається ще один можливо вагомий фактор – це помста гравця за не співпрацю у попередніх іграх. Рівновагою гри може стати співпраця, але іноді бажання зрадити іншу людину може бути сильнішим за загрозу покарання. При

зростанні числа повторювань гри рівновага Неша починає прямувати до Парето-оптимуму.

Дилема ув'язненого має наступний класичний вигляд: суб’єкти А та Б, обидва з них є арештованими підозрюваними. У правоохоронних органів недостатня кількість доказів, щоб впевнено стверджувати про їх провину. Поліція приймає рішення ізолювати підозрюваних у відокремлені кімнати та пропонує обом ті ж самі варіанти подальших дій: при умові що один дає показання проти другого, а той зберігає мовчання, то перший виходить на волю, а інша людина буде нести покарання у в'язниці на протязі 10 років. Випадок при якому обоє заарештованих не свідчать проти іншого приведе до 6 місяців засудження, через недостатню кількість доказів у поліції. А якщо обидва підозрюваних будуть давати показання один проти одного, то вони отримають засудження на протязі двох років. У ув'язнених є всього два варіанти дій - свідчити проти іншого або мовчати. Момент що ускладнює прийняття рішення є відсутність інформації про вибір іншої людини.

Проблема під час прийняття рішення з’являється при умові, що ув’язнені піклуються в першу чергу за себе – як зменшити свій термін у неволі.

Змоделюємо цю ситуацію – хід думок заарештованих. При умові що спільник ув’язненого буде мовчати, то злочинець може вийти на волю, розповівши про його спільника, або отримає півроку неволі якщо буде збереже мовчання. При умові що спільник розповість про злочин свого партнера, то для злочинця будуть наступні наслідки його дій: мовчати та отримати 10 років в’язниці, або теж свідчити та отримати тільки 2 роки. Як видно з прикладу, відштовхуючись від двох умов що покривають усі варіанти, то для злочинця є вигіднішим свідчити проти свого спільника, а ніж мовчати. Таким самим чином може думати і сам спільник цього ув’язненого, якого щойно було розглянуто, - висновок є тим самим.

Насправді ж, для цих заарештованих найкращим рішенням у цій ситуації є прийняти позицію мовчання. Ця співпраця приведе до тільки пів

року неволі для кожного з них. Що є найменшим сумарним терміном ув'язнення. Перебір усіх варіантів цієї задачі доводить це. Інші варіанти є не настільки вигідними.

Приклад гри «дилема в'язня» добре демонструє ситуацію гри з ненульовою сумою, для якої Парето-оптимальне рішення може бути відмінним (протилежним) від рішення рівноваги Неша.

## 1.3 Метод множників Лагранжа

Метод множників Лагранжа - метод знаходження умовного екстремуму функції $f(x)$, де $x$ є $R^n$, щодо $m$ обмежень $\varphi_i(x) = 0$, де $i$ змінюється від одиниці до $m$. Цей метод може бути застосований для вирішення задач математичного програмування (зокрема, лінійного програмування) [5].

Складемо функцію Лагранжа у вигляді лінійної комбінації функції $f$ і функцій $\varphi_i$, взятих з коефіцієнтами, що мають назву множники Лагранжа - $\lambda_i$:

$$L(x, \lambda) = F_x(x, y) + \sum_{i=1}^{m} \lambda_i * \varphi_i (x_i), \quad (1.4)$$

де $\lambda = (\lambda_1, .., \lambda_m)$

Складемо систему з $n + m$ рівнянь, прирівнявши до нуля частинні похідні функції Лагранжа $L$ (x, $\lambda$) по $x_j$ та $\lambda_i$.

Якщо отримана система має рішення щодо параметрів $x'_j$ та $\lambda_i'$, тоді точка $x'$ може бути умовним екстремумом, тобто рішенням вихідної задачі. Зауважимо, що ця умова носить необхідний, але не достатній характер.

## 1.4 Розбиття чисел

Розбиттям числа ***n*** називають сукупність цілих додатних чисел, сума яких має дорівнювати ***n***. Такі числа, що входять до однієї з цих сукупностей, називають частинами цього розбиття. При цьому не є важливим послідовність частин у сукупностях. Тобто якщо у дві сукупності входять ті ж самі числа, але послідовність чисел відрізняється, то такі сукупності можна вважати рівними.

Позначення ***p(n)*** в теорії чисел співвідносять із кількістю розбиттів натурального числа ***n*** на можливі частини [6].

Наприклад, {1, 1, 2} або {3, 1} — є розбиттями числа 4, оскільки 4 = 1 + 1 + 2 = 3 + 1. Всього існує ***p(4)***=5 розбиттів числа 4: {1, 1, 1, 1}, {2, 1, 1}, {2, 2}, {3, 1}, {4}.

**Алгоритм *O($n^3$).*** Нехай ***P(n, m, k)*** - кількість розбиттів числа ***n*** на ***m*** доданків, кожне з яких не перевищує ***k***. Має місце наступне рекурентне співвідношення:

$$P(n,m,k) = \begin{cases} P(n,m,k-1) + P(n-k,m-1,k), & 0 < \mathrm{m} \leqslant \mathrm{n}, 0 < \mathrm{k} \leqslant \mathrm{n} \\ P(n,m,n), & k > n \\ 1, & \mathrm{n} = 0, \mathrm{m} = 0 \\ 0, & \text{інакше} \end{cases}$$

Розглянемо безліч варіантів розбиття числа ***n*** на ***m*** доданків, кожне з яких не більше ***k***. Розділимо його на дві непересічні групи - в першій будуть всі розбиття, які не містять в якості старшого доданка ***k***. Таких варіантів розбиття ***P(n, m, k-1)***. У другій – всі розбиття зі старшим доданком ***k***. Їх стільки ж, скільки варіантів розбиття числа ***n - k*** на ***m - 1*** доданок, кожне з яких не більше за ***k***, тобто ***P(n-k, m-1, k)***.

Кількість всіх варіантів розбиття числа дорівнює $\sum_{i=0}^{\mathrm{n}} \boldsymbol{P}(\boldsymbol{n}, \boldsymbol{i}, \boldsymbol{n})$. Реалізація даного алгоритму методом динамічного програмування з меморізацією матиме асимптотику *O($n^3$)*.

**Алгоритм $O(n^2)$.** Позначимо ***P(n, k)*** як кількість варіантів розбиття числа ***n*** на складові, кожне з яких є не більшим за ***k***. Воно задовольняє наступній рекурентній формулі:

$$P(n,k) = \begin{cases} P(n,k-1) + P(n-k,k), & 0 < \mathrm{k} \leqslant \mathrm{n} \\ P(n,n), & k > n \\ 1, & \mathrm{n} = 0, k = 0 \\ 0, & \mathrm{n} \neq 0, k = 0 \end{cases} \quad (1.5)$$

Зауважимо, що нам не потрібно рахувати кількість доданків ***m*** у розбитті. Досить порахувати ***P(n, k)*** - кількість розбиттів числа ***n*** на довільну кількість доданків, кожне з яких не більше ***k***. Розглянемо безліч таких варіантів розбиття. Розділимо його на дві непересічні групи. До першої увійдуть ті розбиття, в яких відсутній доданок k. Очевидно, таких варіантів розбиття ***P(n, k-1)***. У другій групі - ті розбиття, в які доданок ***k*** увійшов. Їх кількість збігається з кількістю варіантів розбиття числа ***n - k*** на складові, кожна з яких є не більшим за ***k***, та дорівнює ***P(n-k, k)***.

Кількість всіх розбиттів числа ***n*** дорівнює ***P(n, n)***.

Асимптотика реалізації алгоритму буде мати складність $O(n^2)$.

## 1.5 Задача пакування рюкзака

Задача пакування рюкзака (ЗПР) - NP-повна задача комбінаторної оптимізації. Свою назву задача отримала від кінцевої мети: укласти максимально можливу кількість цінних речей в рюкзак за умови, що місткість рюкзака обмежена. З різними варіаціями задачі про рюкзак можна зіткнутися в таких сферах, як економіка, прикладна математика, криптографія та логістика.

У загальному вигляді задачу можна сформулювати так: із заданої множини предметів з такими властивостями, як «вартість» і «вага» потрібно відібрати підмножину з максимально можливою загальною цінністю,

дотримуючись при цьому обмеження на сумарну вагу таких речей, що можливо розмістити у ранці.

Основу задачі пакування рюкзака можна застосувати під час моделювання, наприклад, наступних проблем:

- під час вибору напрямку для інвестицій потрібно знайти ефективний баланс між ризиком та прибутком від вкладених ресурсів. Такі системи допомагають керувати балансом та диверсифікацією активів особи або організації;
- під час завантаження транспортного засобу, такого як човен чи літак: відбувається пошук такої підмножини серед усіх багажів, що надасть можливість найефективнішого завантаження транспортного засобу;
- під час використання матеріалу, що потрібно ділити на частини (тканини, сталеві листки тощо): оптимальна схема розподілу дозволить використати матеріал із найменшою кількістю відходів.

Вважається, що ця задача отримала своє походження з ситуації, при якій турист має вирішити які з речей він зможе взяти з собою у подорож: очевидно що йому краще обрати речі першої необхідності та при цьому їх обсяг має бути у межах того рюкзака з яким буде подорожувати.

На допомогу визначення задачі пакування рюкзака може прийти математичний апарат. Кожному об’єкту, що може бути покладеним у рюкзак, можна зіставити індекс $i$, який знаходиться у межах від 1 до $n$. Значення $w_i$ та $p_i$ – є вагою та вартістю об’єкта $i$. А ось максимальна вантажопідйомність рюкзака дорівнює $W$.

Одним з варіантів рішення задачі заповнення рюкзака є використання двійкових значень, тобто нуля або одиниці. Такі значення будуть являтися індикаторами того, що певна $i$-та річ була покладена у рюкзак. А якщо всі речі, що мають шанси потрапити у рюкзак, упорядкувати за $i$ від 1 до $n$, то ми можемо отримати наступний двійковий вектор $X = (x_1, x_2, \ldots, x_n)$, де $x_i$ приймає значення одиниці у випадку що об’єкт потрапить до рюкзаку, а нуль – не потрапить. Такий вектор має наступну назву - вектор заповнення

рюкзака речами. Маючи цей вектор можна знайти наскільки є заповненим рюкзак зараз, та якою є загальна вартість речей, що вже були покладені.

Поточна завантаженість рюкзака, або загальна вага покладених предметів для вектору $X$ обчислюється за такою формулою:

$$w(X) = \sum_{i=0}^{n} x_i \, w_i \quad (1.6).$$

А цінність предметів обчислюється наступним чином:

$$z(X) = \sum_{i=0}^{n} x_i \, p_i \quad (1.7).$$

Відштовхуючись від цих формул задачу пакування рюкзака для вектору $X$ можна переписати так: потрібно максимізувати функцію $z(X)$ при умові що $w(X) < W_{\text{р}}$, де $W_{\text{р}}$ – верхня межа ваги предметів у рюкзаку.

Серед додаткових умов до цієї задачі, що випливають із умов вище, можна виділити наступні:

- одночасно не можна покласти разом усі предмети, що розглядаються, у рюкзак;
- для усіх значень $i$ від 1 до $n$, значення $p_i$ та $w_i$ є строго більшими за нуль ( $p_i > 0$ та $w_i > 0$). Тобто предмети, що розглядаються, мають певну не нульову цінність та вагу.

В якості деяких важливих термінів такої задачі можна виділити наступні:

- Цільовою функцією задачі є $z(X)$, яку в залежності від задачі максимізують (зазвичай) або мінімізують;
- Вектор $X$ є припустимим, якщо $w(X) < W_{\text{р}}$. Це означає що такий вектор буде входити у підмножину можливих оптимальних рішень цієї задачі;
- Вектор $X$ є оптимальним, якщо $z(X)$ – є максимальною, якщо порівнювати значення для будь-якого іншого $x_i$.

## 1.6 Динамічне програмування

Одним із методів, за допомогою яких є можливість знайти точне рішення задачі пакування рюкзака є динамічне програмування (ДП).

У задачі пакування рюкзака проглядається можливість знаходження рішення задачі з *k* змінними, що буде оптимальним, відштовхуючись від рішення на основі задачі з *k-1* змінними. Така властивість задачі вказує на те, що ЗПР має властиву до суб-оптимальної структуру. А саме це і дозволяє використовувати динамічне програмування в якості можливого методу знаходження рішення цієї задачі [7].

Припустимо що перед нами стоїть задача знайти рішення задачі пакування 0-1 (що описана у пункті 1.5). Зробимо наступне позначення *KP(i, c),* значення якого дорівнює максимальній вартості перших *i* предметів та при цьому вага таких предметів не перевищує *c*. А кінцевим результатом треба знайти *KP(n, W)*.

Для пошуку значення *KP(i, c)* може прийти в нагоду наступна ідея: розділити головну задачу на дві більш простіші. Першою буде задача з *i-1* змінними для рюкзака з тією самою ємністю та *KP(i−1, c)*, та $x_i$=0. Другою буде задача також з *i-1* змінними для рюкзака, але ємність дорівнює $c - w_i$ та *KP(i−1,* $c - w_i$*)*, та $x_i$=1.

При цьому логічними є твердження про те, що при нульовій кількості змінних (*KP(0, *)*) або при нульовій ємності рюкзака (*KP(*, 0)*) значеннями максимальної вартості (*KP)* будуть нулі.

Згідно з цими даними тепер є можливість рекурсивно знайти *KP(i, c)*:

- *KP(0, c)= 0, при* $0 \le c \le W$,
- *KP(i, 0)=0, при* $0 \le i \le n$,
- *KP(i, c)= KP(i-1, c), якщо* $w_i > c$ *– i-ий* предмет є важчим за поточні можливості пакування рюкзака,
- *KP(i, c)=max(KP(i-1, c), KP(i-1,* $c - w_i$*)) якщо* $w_i \le c$.

Після цього ми вже маємо можливість побудувати таблицю T[i, c]. Її елементи – це значення *KP(i, c)*, які можна знайти за допомогою програмного коду, подібного на цей:

```
for c from 0 to W do:
   T[0, c] := 0
for i from 1 to n do:
   for j from 0 to W do:
      if w[i] > j then:
         T[i, j] := T[i-1, j]
      else:
         T[i, j] := max(T[i-1, j], T[i-1, j-w[i]] + p[i])
```

Складністю по часу виконання такого алгоритму дорівнює *O(nW)*.

Звісно ж, всі алгоритми мають свої переваги та недоліки. До переваг входять: швидкість обчислювань та відсутність необхідності у сортуванні змінних.

До недоліків можна віднести потребу у порівняно великому обсязі пам’яті. А це є великим мінусом для задач з великим обсягом змінних.

## Висновки до розділу 1

У цьому розділі було розглянуто той теоретичний матеріал, без якісного розуміння якого було б неможливим подальший аналіз літератури та самі дослідження. Серед яких є матеріал із теорії ігор, теорії чисел, рефлексивного аналізу, теорії керування та обчислювальних систем. Використання такого переліку матеріалу за напрямком роботи вказує на складність теми, що розглядається.

Теорія ігор в цій роботі є термінологічним базисом, від якого будуть відштовхуватися та який іноді будуть використовувати інші методи.

Дилема в’язня тут зазначена в якості прикладу, на фоні якого можна порівняти складність знаходження розв’язку для ігор ГПБ.

Підрозділ про розбиття чисел дає нам змогу оцінити складність обчислень для стандартного методу пошуку рішення у грі Блотто.

Метод множників Лагранжа, як буде показано нижче, стане в нагоді при пошуку рівноваги Неша у ГПБ.

Розглянуті деякі з методів, що застосовуються для рішення задач математичного програмування. Вони будуть застосовані під час пошуку множин із оптимальними рішеннями задачі.

# 2 ГРА ПОЛКОВНИКА БЛОТТО

Гра полковника Блотто (ГПБ) є однією з класичних задач теорії ігор. Ця гра відбувається між двома особами, що конкурують між собою у битві за певну кількість об’єктів (поля поєдинків). На кожному полі сторона, яка розгорнула найбільше військ, виграє бій. Полковник, який найбільш ефективно розподілить свої війська та виграє більшість боїв - виграє гру.

Гра Блотто має доволі широкий потенціал доцільності вибору цієї моделі, коли настає потреба розділу ресурсів по декільком напрямкам, що може включати у себе бізнес, логістику та політичні кампанії. Проте, навіть простий варіант гри може містить в собі мільйони можливих стратегій, і важко або неможливо вивести найкращу стратегію на папері за адекватний час.

## 2.1 Загальна модель ГПБ

Наведемо більш чітке визначення гри полковника Блотто. Існує кілька різних "канонічних" форм цієї гри Блотто, деякі з яких є іграми з нульовою сумою, а деякі з них не є такими. Гравці одноразово, одночасно і незалежно (не знаючи вибору опонента) розподіляють свої обмежені ресурси між кінцевим числом об'єктів (полів битв або об'єктів захисту/нападу, одночасних конкурсів/аукціонів, груп виборців і т.п.). Переможе той гравець (полковник), що розмістить свої ресурси найбільш ефективно та переможе у більшості об’єктів. Дана модель є канонічним прикладом застосування теорії ігор до військової справи, та ще є однією із перших що були застосовані у цій сфері [8].

Позначимо через $n$ – кількість об'єктів, через $x = (x_1, ..., x_n)$ – дія/стратегія першого гравця (вектор розподілу ресурсів), через $y = (y_1, ..., y_n)$

- дія другого гравця, де $x_i \geq 0$ $(y_i \geq 0)$ - кількість ресурсу, виділеного першим (другим) гравцем на $i$-ий об'єкт, де $i$ від 1 до $n$. Обмеженість ресурсів відображена умовами:

$$\sum_{i=1}^{n} x_i \leq R_x, \quad \sum_{i=1}^{n} y_i \leq R_y \tag{2.1}$$

## 2.2 Аукціонна модель

В рамках **аукціонної моделі** перемогу на об'єкті здобуває гравець, який виділив на нього більшу кількість ресурсів (в разі рівності ресурсів кожен з гравців здобуває перемогу з ймовірністю 1/2). Цінність отримання $i$-го об'єкта для першого (другого) гравця позначимо через $X_i$ ($Y_i$). Тоді функція виграшу першого гравця для одного $i$-го об'єкту визначається як:

$$f_{i\,x}(x_i, y_i) = X_i * p(x_i, y_i), \tag{2.2}$$

$$\text{де } p(x_i, y_i) = \begin{cases} 1, & \text{якщо } x_i > y_i \\ \frac{1}{2}, & \text{якщо } x_i = y_i \\ 0, & \text{якщо } x_i < y_i \end{cases}. \tag{2.3}$$

Функція виграшу для другого гравця для одного $i$-го об'єкту:

$$f_{i\,y}(y_i, x_i) = Y_i * p(y_i, x_i), \qquad \text{де } p(y_i, x_i) = \begin{cases} 1, & \text{якщо } y_i > x_i \\ \frac{1}{2}, & \text{якщо } y_i = x_i \\ 0, & \text{якщо } y_i < x_i \end{cases}. \tag{2.4}$$

Тобто звідси є можливість отримати висновок про існування наступної рівності: $p(y_i, x_i) = 1 - p(x_i, y_i)$.

А загальні виграші гравців для всієї гри в аукціонної моделі будуть визначатися в такий спосіб:

$$f_x(x, y) = \sum_{i=1}^{n} f_{i\,x}(x_i, y_i) = \sum_{i=1}^{n} X_i * I(x_i > y_i) + \frac{1}{2} \sum_{i=1}^{n} X_i * I(x_i = y_i) \tag{2.5}$$

$$f_y(x,y) = \sum_{i=1}^{n} f_{i\,y}(y_i, x_i) = \sum_{i=1}^{n} Y_i * I(y_i > x_i) + \frac{1}{2}\sum_{i=1}^{n} Y_i * I(x_i = y_i) \qquad (2.6)$$

Тут $I$(*) - функція-індикатор. Більш загальним є випадок, коли обмеження типу (2.1) відсутні, але з виграшу (2.5 або 2.6) віднімаються витрати, монотонні за сумарною кількістю використаного гравцем ресурсу.

У одних із перших варіацій ГПБ не розглядалася можливість того, що перемога на певному з об'єктів для одного гравця може принести $X_i$, а для іншого $Y_i$, або те що взагалі виграш від перемоги на одному об'єкті може якось відрізнятися від виграшу при перемозі на іншому об'єкті (полі). У самому першому прикладі гри полковника Блотто, що було описано у літературі теорії гри Борелем у 1921 році, всі об'єкти були рівноцінні для суперників. Та людина що переможе на більшості полів – переможе у грі. Вже тоді Борель вперше описав клас ігор, у яких використовується прийняття рішень щодо розподілу ресурсів між кінцевою кількістю полів. [9]

Випадки $n = 1$ і $n = 2$ є тривіальними. Дійсно, при $n = 1$ перемагає гравець, що володіє більшою кількістю ресурсу (в разі рівності ресурсів перемога кожного рівноймовірна). При $n = 2$ оптимальною стратегією кожного гравця є пріоритетне виділення ресурсу на найбільш цінний для нього об'єкт.

Через три десятиліття після опису гри Борелем у 1950 році вона була вдосконалена у роботі Гросса та Вагнера, що тоді працювали у RAND Corporation. Саме вони назвали гру ім'ям вигаданого полковника – «Гра полковника Блотто». Після цього і до нині назва гри збереглася [10].

Найпростішим є симетричний ($X_i = Y_i$, $i \in N$, $R_x = R_y$) варіант дискретної ГПБ (ресурси гравців дискретні), що є матричною грою (з нульовою сумою). Вперше рішення (рівновага Неша в змішаних стратегіях) цієї гри для випадку n = 3 було описано в [11], Потім були знайдені рішення для симетричного випадку для довільного кінцевого n і для випадку $X_i = Y_i$, $i \in N$, $R_x \neq R_y$ при n =2. Наступним кроком була часткова характеризація рівноваги Неша для

випадку $X_i = Y_i$, і є $N$, $R_x \neq R_y$ при довільному кінцевому n. Надалі, як правило дослідники обмежувалися або дискретним, або симетричним безперервним випадками. Істотне просування в характеризації аукціонної моделі було отримано в [12] та [13], де характеризується рівновагу Неша в чистих стратегіях для несиметричного випадку.

Хоча ГПБ була представлена у літературі дуже рано на початку розвитку теорії ігор, та навіть раніше за усім відому гру «Дилема ув'язненого», не дивлячись на це гра полковника Блотто отримала менше уваги серед дослідників ніж добре відома дилема. Таке ставлення могло виникнути по причині заплутаності гри Блотто та складності знаходження її розв’язків. На відміну від найпростішої форми гри «Дилема ув'язненого» (підрозділ 1.2), яка містить лише дві стратегії та простий аналіз матриці виплат 2х2, навіть дуже прості версії гри полковника Блотто мають набагато складніші простори стратегій. У таблиці 2.1 показано кількість можливих стратегій для деяких вибраних простих версій гри Блотто.

Таблиця 2.1 - Кількість стратегій гри Блотто при заданих параметрах

| Кількість одиниць ресурсів | Кіл-сть об’єктів | Кіл-сть стратегій |
|---|---|---|
| 6 | 2 | 7 |
| 5 | 4 | 56 |
| 12 | 3 | 91 |
| 24 | 4 | 2925 |
| 50 | 5 | 316251 |
| 120 | 6 | більше ніж 250 млн |

Ця таблиця 2.1 ярко відображає ту складність обчислень, яка може виникати під час аналізу всіх стратегій гри, їх порівняння та знаходження оптимального рішення. Число можливих унікальних стратегій росте з

факторіальною швидкістю з ростом кількості об’єктів/полів. Через це, навіть ігри з відносно невеликими кількостями одиниць ресурсів та об’єктів можуть породжувати матриці виплат із багатьма тисячами рядків та стовбців. Крім того, Роберсон (2006), Тофіас (2007) та інші довели, що гра Блотто не має рівноваги Неша в чистих стратегіях. З огляду на вектор корисності об’єктів та стратегію противника (вектор розміщення ресурсів), гравець завжди може змінити порядок призначення своїх військ, щоб покращити свій рахунок і виграти гру (при умові порівнюваної кількості ресурсів кожного з гравців).

У порівнянні з простотою гри «Дилема ув'язненого» та легкістю знаходження її рівноваги Неша, стає зрозумілим чому складна гра Блотто у теорії ігор була відсторонена від уваги дослідників впродовж багатьох років.

По факту, задача знаходження конкретних стратегій, що були б рівновагами Неша, та стани гри Блотто залишалися невирішеними протягом 85 років після першого опису гри Борелем, і часто розглядалися як нерозв'язні для безперервного чи дискретного випадку. Однак великий вклад у цей напрямок було зроблено у 2006 році Роберсоном. Він вдало виявив рівноважні стратегії розподілу для безперервних ігор Блотто будь-якого розміру, як функції від відносної корисності даного поля та загальної кількості наявних військ.

Висновки Роберсона викликали новий інтерес до гри Блотто, особливо в нещодавно популярній галузі експериментальної економіки. У багатьох роботах з'ясовано потенціал експериментального аналізу гри Блотто і створені дослідження, в яких закликають студентів, науковців та фахівців з бізнесу, щоб вони запропонували свої стратегії, які можна було б використовувати в турнірах ігор Блотто.

## 2.3 Ймовірнісна модель гри

У ймовірнісної моделі ГПБ ймовірність $p_x(x_i, y_i)$ перемоги першого гравця на $i$-му об'єкті не залежить від інших об'єктів і «пропорційна» кількості виділеного їм на цей об'єкт ресурсу та «зворотно пропорційна» зваженої сумі ресурсів, виділених на цей об'єкт обома гравцями:

$$p_x(x_i, y_i) = \frac{\alpha_i * x_i}{\alpha_i * x_i + y_i},\ p_y(x_i, y_i) = 1 - p_x(x_i, y_i) \tag{2.7}$$

та $p_{x,y}(x_i = 0, y_i = 0) = \frac{\alpha_i}{\alpha_i + 1}$,

де $\alpha_i > 0$ – відносна ефективність розташування першим гравцем одиниці ресурсів до одиниці ресурсів суперника на $i$-ому об’єкті.

Виграші гравців в ймовірнісної моделі визначаються як математичне очікування сумарного виграшу, тобто в такий спосіб:

$$F_x(x, y) = \sum_{\mathrm{i}=1}^{n} X_\mathrm{i} * p_x(x_i, y_i)\,, F_y(x, y) = \sum_{\mathrm{i}=1}^{n} Y_\mathrm{i} * p_y(x_i, y_i) \tag{2.8}$$

Рівновагою Неша в чистих стратегіях ($x^*$, $y^*$) є пара векторів, що задовольняють умовам (2.1), таких, що $V(x, y)$, які відповідають умовам (2.1), виконано:

$$F_x(x^*, y^*) \geq F_x(x, y^*), F_y(x^*, y^*) \geq F_y(x^*, y) \tag{2.9}$$

Ймовірнісна модель в певному сенсі «простіше» ніж аукціонна, єдиною рівновагою Неша для випадку $X_i = Y_i = \text{const}$, $i \in N$, $R_x \neq R_y$ є використання гравцями чистих стратегій, які полягають в рівному розподілі наявних у них ресурсів між об'єктами.

$$x_i^* = \frac{V_i}{V} R_x,\ \ y_i^* = \frac{V_i}{V} R_y.\ \mathrm{i} \in \mathrm{N} \tag{2.10}$$

$$F_x(x^*, y^*) = \frac{R_x}{R_x + R_y} V\ ,\ \ F_y(x^*, y^*) = \frac{R_y}{R_x + R_y} V \tag{2.11}$$

де $V = \sum_{\mathrm{i}=1}^{n} V_\mathrm{i}$ , тобто гравці ділять свій ресурс пропорційно цінності об'єктів і отримують виграш, пропорційний їх сумарним ресурсам. Відзначимо, що при цьому рівноважні дії кожного з гравців залежать тільки від «їх власних»

параметрів - так, наприклад, дії першого гравця $x^*$ не залежать від сумарної кількості ресурсу $R_y$ у другого гравця і т.п.

## 2.4 Пошук рівноваги Неша

У [14] було знайдено рівновагу в чистих стратегіях для випадків $X_i = Y_i$, для довільних $\alpha_i > 0$, $i \in N$. Спробуємо знайти розв'язок не враховуючи спрощення $X_i = Y_i$.

У ході гри ми маємо максимізувати функцію виграшу $F_x(x, \mathrm{y})$ для першого гравця. При цьому ми маємо декілька обмежень щодо області з якої можна вибирати значення $x_i$, $i \in N$. Обмеження можна позначити як $\varphi_i(x_i)$, де $i$ знаходиться в межах від 0 до $m$ – кількість обмежень, при цьому їх потрібно звести до нуля, шляхом перенесення правої частини нерівності обмеження у ліву зі знаком мінус. Таким чином тепер можна використати метод множників Лагранжа (підрозділ 1.3), що застосовується для знаходження розв'язків математичного програмування (лінійного), через пошук умовних екстремумів функції що потрібно максимізувати (мінімізувати).

$$L(x, \lambda) = F_x(x, \mathrm{y}) + \sum_{i=1}^{m} \lambda_i * \varphi_i\,(x_i) \tag{2.12}$$

Розглянемо випадок ймовірністної моделі, із наступними функціями:

$$F_x(x, \mathrm{y}) = \sum_{\mathrm{i}=1}^{n} X_{\mathrm{i}} * p_x(x_i, y_i) = \sum_{\mathrm{i}=1}^{n} X_{\mathrm{i}} * \frac{\alpha_i * x_i}{\alpha_i * x_i + y_i} \tag{2.13}$$

$$\varphi_i\,(x_i) = R_x - \textstyle\sum_{\mathrm{i}=1}^{n} x_{\mathrm{i}}$$

Знайдемо часткову похідну функції $L(x, \lambda)$ по $x_i$:

$$\frac{\partial L}{\partial x_i} = \frac{\partial\left(\sum_{j=1}^{n} X_j * \frac{\alpha_j * x_j}{\alpha_j * x_j + y_j}\right)}{\partial x_i} + \frac{\lambda_i * \partial\left(R_x - \sum_{i=1}^{n} x_i\right)}{\partial x_i} = \tag{2.14}$$

$$= \frac{\partial\left(X_i * \frac{\alpha_i * x_i}{\alpha_i * x_i + y_i}\right)}{\partial x_i} - \lambda_i * \frac{\partial x_i}{\partial x_i} = \frac{X_i * \alpha_i * y_i}{\left(\alpha_i * x_i^* + y_i^*\right)^2} - \lambda_i = 0$$

Звідси виходить що множник Лагранжа для першого гравця має наступний вигляд:

$$\lambda_x = \frac{\alpha_i * y_i^* * X_i}{(\alpha_i * x_i^* + y_i^*)^2} \tag{2.15}$$

Аналогічним чином можна знайти $\lambda_y$:

$$\lambda_y = \frac{\alpha_i * x_i^* * Y_i}{(\alpha_i * x_i^* + y_i^*)^2}. \tag{2.16}$$

Розділивши $\lambda_x$ на $\lambda_y$ отримаємо наступне відношення:

$$\frac{\lambda_x}{\lambda_y} = \frac{y_i^*}{x_i^*} * \frac{X_i}{Y_i} \tag{2.17}$$

Звідки можна виразити відношення $x_i^*$ до $y_i^*$:

$$\frac{x_i^*}{y_i^*} = \frac{\lambda_y}{\lambda_x} * \frac{X_i}{Y_i} \tag{2.18}$$

Використовуючи (2.18) тепер є можливість виразити $x_i^*$ та $y_i^*$ через $\lambda_x$, $\lambda_y$:

$$x_i^* = \frac{\lambda_y}{\lambda_x} * \frac{X_i}{Y_i} * y_i^*, \tag{2.19}$$

$$y_i^* = \frac{\lambda_x}{\lambda_y} * \frac{Y_i}{X_i} * x_i^* \tag{2.20}$$

Для випадку $X_i$=$Y_i$ (2.19) або (2.20) можна виконати підсумовування по всім $i$:

$$R_x = \sum_{j=1}^{n} x_i^* = \frac{\lambda_y}{\lambda_x} * R_y \tag{2.21}$$

Звідки ми отримуємо наступне відношення:

$$\frac{\lambda_x}{\lambda_y} = \frac{R_y}{R_x} \tag{2.22}$$

Що спрощує (2.19) та (2.20):

$$x_i^* = \frac{R_x}{R_y} * y_i^* \text{ та } y_i^* = \frac{R_y}{R_x} * x_i^* \quad (2.23)$$

## 2.5 Стратегічна рефлексія у грі Блотто

З точки зору теорії ігор і **рефлексивних моделей** прийняття рішень виділяють стратегічну і інформаційну рефлексію [15].

Інформаційна рефлексія - процес і результат роздумів гравця про те, які значення невизначених параметрів, що про ці значеннях знають і думають його опоненти (інші гравці). При цьому власне «ігрова» компонента відсутня, так як ніяких рішень гравець не приймає.

Іншими словами, інформаційна рефлексія відноситься до інформованості гравця про природної реальності (яка гра) і про рефлексивної реальності (якою бачать гру інші). Інформаційна рефлексія логічно передує рефлексії дещо іншого роду - стратегічної рефлексії.

Стратегічна рефлексія - процес і результат роздумів гравця про те, які принципи прийняття рішень використовують його опоненти (інші гравці) в рамках тієї інформованості, яку він їм приписує в результаті інформаційної рефлексії.

Таким чином, інформаційна рефлексія має місце тільки в умовах неповної інформованості, і її результат використовується при прийнятті рішень (в тому числі при стратегічної рефлексії). Стратегічна рефлексія має місце навіть у разі повної інформованості, випереджаючи прийняття гравцем рішення про вибір дії. Іншими словами, інформаційна та стратегічна рефлексії можуть вивчатися незалежно, проте в умовах неповної інформованості обидві вони мають місце.

Розглянемо аспекти стратегічної рефлексії для ГПБ. Нехай $X_i = Y_i = V_i$, $i \epsilon N$, тоді рівноважні дії гравців і їх виграші в рамках ймовірнісної моделі

визначаються виразами (2.10) і (2.11) відповідно. Використання концепції рівноваги Неша, як прогнозованого сталого результату некооперативного гри, має на увазі, що параметри гри є загальним знанням [15]. ГПБ в рамках ймовірнісної моделі описується, по-перше, кортежем ($N$, $R_x$, $R_y$, $\{V_i\}$), які мають масу об'єктів, обмеження на ресурси гравців і цінності об'єктів для гравців. По-друге, необхідно задати «правила гри» - ймовірності виграшу (2.7) і цільові функції (2.8), прагнення до максимізації яких відображає раціональність поведінки гравців. Умовно можна вважати, що інформаційна рефлексія відповідає відсутності загального знання щодо кількостей ресурсів гравців і цінностей для них об'єктів, а стратегічна рефлексія - щодо принципів прийняття гравцями рішень.

З виразу (2.10) випливає, що, зокрема, якщо $R_x > R_y$, то $Vi \in N$ виконано $x_i^* < y_i^*$, тобто за критерієм (2.5) перший гравець програє другому гравцеві на всіх об'єктах. Раціональний гравець при цьому може задуматися, чи правильно він діє, і, можливо, перегляне свої принципи прийняття рішень.

Позначимо через $BR_x\ (y) = (u_1y_1 + \alpha, \ldots, u_ny_n + \alpha)$ - вектор найкращої в сенсі критерію (2.5) відповіді першого гравця на вибір другим гравцем вектора дій $y$, де n-мірний вектор $u = (u_1, \ldots, u_n)$ є рішенням наступної задачі про ранці:

$$\begin{cases} \sum_{\mathrm{i}=1}^{n} u_i * V_{\mathrm{i}} \to max_{u_i \in \{0,1\}}\ , \\ \sum_{\mathrm{i}=1}^{n} u_i * y_i \ \le \mathrm{R}_x \end{cases} \tag{2.24}$$

де $\alpha = \frac{1}{n} * (\mathrm{R}_x - \sum_{\mathrm{i}=1}^{n} u_i * y_i)$, тобто будемо вважати, що гравець прагне перемогти на найбільш цінному для нього (в рамках ресурсних обмежень) наборі об'єктів, а залишок ресурсу розподіляє порівну між усіма об'єктами.

Аналогічно введемо $BR_y\ (x) = (v_1x_1 + \beta, \ldots, v_nx_n + \beta)$ - вектор найкращої в сенсі критерію (2.5) відповіді другого гравця на вибір першим гравцем

вектора дій $x$, де n-мірний вектор $v = (v_1, ..., v_n)$ є рішенням наступної задачі про ранці:

$$\begin{cases} \sum_{i=1}^{n} v_i * V_i \rightarrow max_{v_i \in \{0,1\}}, \\ \sum_{i=1}^{n} v_i * x_i \leq R_y, \end{cases} \tag{2.25}$$

де $\beta = \frac{1}{n} * (R_y - \sum_{i=1}^{n} v_i * x_i)$.

Рівновага Неша в аукціонної моделі ГПБ може будуватися і досліджуватися за допомогою аналізу властивостей відображень найкращих відповідей $BR_x(*)$ і $BR_y(*)$. Однак нас будуть цікавити ефекти стратегічної рефлексії. Для їх відображення припустимо, що гравці які не рефлексують вибирають рівновагу Неша, відповідне ймовірнісної моделі (2.10) ГПБ. Гравець першого рангу рефлексії вибирає свої дії як найкращу в сенсі (2.24) або (2.25) відповідь на дії опонента, що не рефлексує, вважаючи, що останній діє в рамках ймовірнісної моделі.

Відповідно до прийнятої в теорії рефлексивних ігор традицією будемо вважати, що гравець, який має певний ранг стратегічної рефлексії, вважає що опонент має ранг на одиницю менше його власного. Тобто, має місце наступний «ланцюжок»:

$$\begin{aligned} &x^1 = BR_x(y^*),\ y^1 = BR_y(x^*), \\ &x^2 = BR_x(y^1) = BR_x(BR_y(x^*)),\ y^2 = BR_y(x^1) = BR_y(BR_x(y^*)), \\ &x^k = BR_x(BR_y(\ldots)),\ y^k = BR_y(BR_x(\ldots)), \end{aligned} \tag{2.26}$$

де $x^k$ $(y^m)$ - дія першого (другого) гравця, який володіє $k$-им ($m$-им) рангом стратегічної рефлексії, $k, m = 1, 2, ....$

Досліджуємо гру рангів [15], в якій перший і другий гравці вибирають не кількості ресурсу, як у вихідній ГПБ, а свої ранги, які відповідно до (2.26) детермінують розподіл ресурсів і, отже, виграші гравців (2.5). У грі рангів перший гравець вибирає свій ранг $k \in \{0, 1, 2, ...\}$, другий гравець - свій ранг $m \in \{0,1,2,...\}$. Кожній парі рангів ставиться у відповідність пара чисел

($f_x(k,m)$, $f_y(k, m)$) - виграшів відповідно першого і другого гравця. Тобто розглянута гра рангів є (в загальному випадку нескінченною) біматричною грою (в розглянутій моделі - грою з постійною сумою). Відзначимо, що досліджені на сьогоднішній день ігри рангів були кінцевими, так як «надбудовувалися» над кінцевими матричними або біматричними іграми.

Для визначеності будемо вважати, що виконано $R_y > R_x$. З урахуванням виразу (2.10) при $x = x^*$, $y = y^*$ системи (2.24) і (2.25) приймуть відповідний вид:

$$\begin{cases} \sum_{i=1}^{n} u_i * V_i \to max_{u_i \in \{0,1\}} \\ \sum_{i=1}^{n} u_i * V_i \le V\frac{R_x}{R_y} \end{cases} \quad (2.27)$$

та

$$\begin{cases} \sum_{i=1}^{n} v_i * V_i \to max_{v_i \in \{0,1\}} \\ \sum_{i=1}^{n} v_i * V_i \le V\frac{R_y}{R_x} \end{cases} \quad (2.28)$$

Як зазначалося вище, другий гравець в рівновазі Неша (2.10) має перевагу на всіх об'єктах. З (2.28) випливає, що найкращою відповіддю другого гравця на фіксовану стратегію першого гравця буде виділяти на кожен об'єкт стільки ж ресурсу, скільки виділив перший, а залишок ресурсу розподіляти, наприклад, порівну між усіма об'єктами (обмеженню в задачі (2.28) задовольняє будь-який вектор, навіть той, що складається з усіх одиниць). При цьому другий гравець буде мати перевагу на всіх об'єктах. Іншими словами, описано у виразі (2.29)

$$\forall\, l \in \{0,1,2,..\}: f_x(l, l+1) = 0, f_y(l, l+1) = V \quad (2.29)$$

## Висновки до розділу 2

У цьому розділі було розглянуто основні види моделей гри полковника Блотто та деякі з інструментів, що використовуються зараз при їх дослідженні та можуть бути використані в цій роботі також.

Аналіз показав, що на сьогоднішній день добре дослідженими моделями є ймовірнісні або ті, у яких гравці мають однакові вектори корисностей перемог на об’єктах. Для таких випадків є можливість знайти рівновагу Неша та просто використати запропоновану таким чином стратегію. Шанси отримання максимального прибутку від вибору такої стратегії будуть вищі, ніж для інших.

Практичних досліджень ГПБ, де були б використані різні корисносні об’єктів для різних гравців, знайдено не було. Існує лише декілька статей у яких були описані деякі рекомендації розташування ресурсів для таких випадків. Деякі з таких результатів описані у підрозділах 3.2 та 4.1.

Цікаві результати були отримані за допомогою стратегічної рефлексії у підрозділі 2.5. Що вказує на сенс у використані деяких кроків запропонованих у ньому при розробці нового методу пошуку оптимальних стратегій.

Наступна ситуація з дослідженнями аукціонних моделей вказує на доцільність у продовжені пошуку нових методів, що будуть запропоновувати оптимальні вектори розміщення ресурсів між об’єктами у грі.

# 3 ДОСЛІДЖЕННЯ ІСНУЮЧИХ МОДЕЛЕЙ ГПБ

## 3.1 Проведення симуляції ГПБ

Одним із перших кроків, що потрібні для проведення симуляції моделі гри Блотто, є вирішення питання про те, як генерувати випадкові вектори стратегій з нуля, які можна використовувати в грі для певного набору параметрів. Такі вектори стратегій мають дві властивості: сума їх значень повинна дорівнювати загальній кількості військ або ресурсів (тобто жодна стратегія не може використовувати більше військ, ніж існує у гравця, і жодна стратегія не може залишити війська невикористаними), та довжина векторів повинна дорівнювати *n*, де *n* - кількість полів у цій грі. Генерація стратегій була однією з перших обчислювальних задач проекту, через існування двох різних підходів про використання справді «випадкового» вектору стратегій.

Один із способів генерування випадкових векторів стратегій - це почати з функції генератора, яка може створювати, по одному за раз, перестановки стратегій довжини *n*, що складають загальну кількість доступних ресурсів. Потім генератор може бути викликаний списком для заповнення цього списку всіма можливими векторами стратегій на просторі стратегій. На мові програмування Python існує функція random.choice(), що може бути використана для вибору стратегії зі списку випадковим чином, якщо це необхідно [16].

Цей підхід має кілька переваг. Створюючи весь простір можливих стратегій і використовуючи random.choice(), ми гарантуємо, що наш вибір випадкового вектора стратегії буде рівномірно "випадковим" по кожній стратегії в просторі (звичайно, ми обмежені обмеженнями "Псевдо-випадковість" при використанні комп'ютерів для генерації випадкових подій). Ми також можемо легко шукати серед усіх стратегій для даного набору параметрів в нашому пошуку, щоб знайти найкращі стратегії для гри Блотто, не побоюючись, що ми могли б пропустити потенційно хорошу

стратегію через випадкову помилку. Також легко створити впорядкований список всіх стратегій, щоб класифікувати ефективність певних стратегій порівняно з усіма іншими.

Проте зручність і ретельність підходу генератора підривається великими втратами ефективності в обчислюваннях, особливо при роботі з великими просторами стратегій. Навіть якщо ми хочемо отримати лише невелику підмножину стратегій, ми повинні генерувати весь простір стратегії, перш ніж витягти бажаний результат. У моделі ГПБ із 6-ьма полями та 120 одиницями ресурсів, подібно до гри Арад-Рубінштейна, це означає, що для створення навіть однієї випадкової стратегії, ми повинні чекати, поки система збере всі 250 мільйонів стратегій у список. Це робить аналіз навіть помірно великих ігор Блотто обчислювально неефективним і вимагає способу генерування випадкових стратегій на вимогу.

Значення координат обирається випадковим чином на проміжку від нуля до кількості ресурсів. Для кожної наступної координати величина проміжку зменшується на величину щойно отриманого випадковим чином значення. Для останньої координати вектору задається значення що є максимальним на проміжку, тобто та кількість ресурсів які ще не були використані.

Ця процедура «на вимогу» для формування стратегій полегшує моделювання турнірів з великими просторами стратегій. Цю процедуру можна викликати будь-яке число разів, щоб генерувати бажану кількість випадкових стратегій, не генеруючи в першу чергу усю множину стратегій. Випадковий характер процесу відбору може також включати деяку стохастичну поведінку до вибору стратегії, яка не існує, коли ми генеруємо весь простір стратегії. Це може бути корисним, якщо припустити, що люди не завжди вибирають найкращі стратегії, а іноді роблять помилки. Отже, створення нашої найкращої стратегії з цією процедурою може бути кращою апроксимацією експериментальних результатів.

Однак, щоб ці результати були надійними, ця процедура не повинна бути упередженою до повернення стратегій з певними характеристиками над іншими. Якщо поглянути на таку процедуру «за вимогою», незрозуміло, чи дійсно вона буде виробляти стратегії, які будуть рівномірно розподілені як процедура генератора. Незважаючи на те, що сукупні результати моделювання від кращих стратегій, створених за допомогою такої процедури, подібні до результатів процедури генератора, тестування виявляє надмірне упереджене ставлення до стратегій, які становлять більшу частину наявних ресурсів в одному полі. Ці стратегії згодом будуть вибрані, коли ми вибираємо найкращі стратегії, так що хоча ця упередженість може впливати на конкретну величину турнірних балів, вона не впливає на загальні результати, які ми представляємо пізніше.

Інакшим чином потрібні нам стратегії можливо генерувати і без звужування проміжку значень на якому випадковим чином буде обиратися значення. Тобто генерація значення для кожної з координат вектору буде проходити незалежно, на відміну від попередньої процедури. Єдиним нюансом буде кінцева перевірка чи дорівнює сума таких чисел повній кількості ресурсів гравця.

Незважаючи на те, що ця процедура не стикається з тим самим зміщенням, як оригінальна представлена спочатку процедура, вона є дуже неефективною, оскільки вона спирається на метод відбору стратегії «вгадати і перевірити». Це робить такий підхід особливо повільним при роботі з великими просторами стратегій. Подальше розширення цього проекту може включати процедуру генерування випадкових стратегій, яка містить рівномірний розподіл між стратегіями у просторі стратегій, до поки обчислювання стануть не ефективними, у разі з передачею параметрів з великими значеннями.

Як тільки ми отримуємо набір стратегій, які ми створили з використанням будь-якого з двох наведених вище методів, ми повинні потім запустити саму імітацію турніру гри Блотто. Для цього потрібно реалізувати

функцію що працюватиме зі стратегіями першого та другого гравця, кількістю об’єктів у грі. Результатом роботи функції буде кількість перемог на об’єктах, або той сумісний виграш, що міг бути отриманий при перемозі на об’єктах. Підрахунок відбувається при по елементному порівнянні значень усіх відповідних координат векторів стратегій гравців: в кого значення більше а ніж у супротивника, той і отримає виграш. При рівних значеннях координат гравці отримають по половині виграшу, інакше – гравець нічого не отримає за розміщення ресурсів (або їх відсутність) на тому об’єкті. Оцінка супротивника може бути отримана шляхом віднімання оцінки стратегії гравця від $n$, загальної кількості полів.

Якщо у нас є список стратегій у вигляді векторів ("набір стратегій"), ми можемо запустити круговий турнір гри Блотто серед усіх стратегій у наборі, просто використовуючи вкладені циклів для запуску гри для кожної стратегії буде відбуватися порівняння з усіма іншими стратегіями у списку, включаючи і саму себе. Список відстежує історію рахунків для стратегій турніру. Потім ми можемо взяти середній бал у списку історії, щоб знайти середній результат турніру стратегій в турнірі. Потім цей результат додається до іншого списку, який може бути перехресно індексований зі стратегічним набором, щоб знайти оцінку турніру для окремої стратегії.

Як тільки ми запустили турнір, корисно відсортувати стратегії для того, щоб їхні бали в турнірі визначили, які стратегії виконані краще. Є можливість створити список стратегій, які сортуються за їхньою продуктивністю в турнірі від найкращого до гіршого.

Тепер, коли наші стратегії відсортовані, може бути бажаним створити список тільки найкращих стратегій і запустити турнір з круговим кроком лише між цими кращими стратегіями. Оскільки експериментальні результати говорять про те, що люди можуть використовувати інтуїцію, щоб уникнути дуже «поганих» стратегій, виконання процесу турніру де будуть порівнюватися тільки найкращі стратегії (однак ми визначаємо, що таке «найкраща стратегія»), ймовірно, буде більш наближеним до

експериментального турніру Блотто. Важливо простежити, чи стратегії, які показують добрі результати поміж усіх стратегій, є такими ж, як стратегії, які показують найкращі результати лише поміж кращими стратегіями (тобто, тільки людськими стратегіями). Цей останній тип стратегій, швидше за все, буде більш прийнятним для експерименту турніру гри Блотто.

Після того, як ми отримали результати турніру, ми можемо повідомити про наші результати. Ця обчислювальна модель повідомляє про стратегії разом з двома турнірними результатами: T-Score і B-Score. T-Score (або Total Score) представляє бал стратегії в турнірі з круглим туром проти репрезентативної вибірки всіх стратегій у просторі. Для невеликих просторів стратегій, T-Score обчислюється шляхом запуску турніру з круглими точками проти всіх стратегій у просторі. Для великих значень параметрів гри T-Score розраховується як оцінка турніру Блотто проти 10 000 випадкових стратегій, які задовольняють параметрам.

B-Score, з іншого боку, є показником стратегії у круглому турнірі проти кращих стратегій. Існують різні способи визначення найкращих стратегій. У простому підході перша третина (наприклад) всіх стратегій, відсортованих за T-Score, класифікується як найкращі стратегії і зберігається у списку. Потім ми можемо запустити турнір з круглим турніром, використовуючи лише стратегії в цьому списку. Середній бал кожної стратегії на цьому турнірі - це B-Score. Диференціація між T-Score і B-Score має на меті сигналізувати про поділ між турнірними показниками в наборі випадкових стратегій і результативністю в наборі стратегій, які б більш нагадували експериментальні результати. Як ми скоро побачимо, стратегії, які добре протиставляються випадковим стратегіям (тобто стратегіям, які мають високий T-Score), не обов'язково добре показують себе у порівнянні з набором найкращих стратегій.

Стратегії також повідомляються разом із їх дисперсією (Var). Var - це стандартний розрахунок дисперсії, який вимірює розсіювання військ між різними полями. Наприклад, стратегія, яка поділяє всі війська на всі наявні

поля, буде мати нульове відхилення. Ми стверджуємо, що люди частіше уникають стратегій з надзвичайно високою дисперсією, тому стратегічні набори, які намагаються імітувати експериментальні результати, також повинні бути зважені до більш низьких відхилень, ніж стратегії з високою дисперсією.

Мета створення стратегічного набору, який би імітував такі стратегії, що могли б бути створені в експериментальних результатах з людськими гравцями, приніс цікаву проблему. Яким буде найкращий спосіб визначити найкращі стратегії з множини випадкових стратегій з великого простору стратегій? Ідеальний набір стратегій міститиме безліч стратегій з низькою дисперсією, в яких змішується декілька стратегій з високою дисперсією, щоб максимально імітувати спосіб, у який люди грають у гру Блотто. Ми можемо створити таку стратегію, як створення агента, який може вивчати стратегії своїх опонентів і зберігати ті стратегії, які добре працюють.

Для цього підходу ми надаємо агенту випадкову стратегію для початку. Потім агент грає ігри Блотто проти інших випадкових стратегій. Якщо така випадкова стратегія агенту є побитою своїм опонентом, то агент відкидає свою поточну стратегію і починає використовувати стратегію свого опонента. Агент також відстежує, скільки ігор виграє певна стратегія поспіль. Якщо стратегія успішно проходить параметр порогової кількості виграних послідовних ігор, перш ніж вона буде відкинутою, стратегія буде додана до списку найкращих стратегій. Агент може продовжувати грати в ігри Блотто, до поки не буде створено заздалегідь визначене число найкращих стратегій з даним критерієм відбору.

Набір стратегій, що виробляється цим агентом який навчається, має кілька бажаних властивостей. Оскільки стратегії з низькою дисперсією, швидше за все, будуть вигравати більше ігор підряд, ніж стратегії з високою дисперсією, набір буде упереджений до стратегій з низьким відхиленням, якщо у нас є суворі критерії відбору. Тим не менш, завжди буде кілька стратегій з високою дисперсією, яким "пощастить" і вони виграють достатню

кількість ігор підряд та будуть додані до набору. Це дає набір, схожий у теорії з експериментальним результатом, отриманим людиною. Крім того, змінюючи силу критерію вибору, ми можемо отримати декілька наборів стратегій з різними пропорціями від стратегій з високою дисперсією до стратегій з низькою дисперсією. Такі маніпуляції виявляться дуже корисними для розкриття того, чому стратегія з високою дисперсією, як (2, 31, 31, 31, 23, 2) виконується так добре в експериментальних турнірах Blotto. Що буде продемонстровано у наступному підрозділі.

## 3.2 Результати проведених досліджень

Чоудхурі разом із іншими дослідниками у 2009 році представили свою статтю з один із перших експериментальних досліджень гри Блотто. У грі в Чоудхурі розглядався асиметричне розподіл ресурсів. Експериментальні результати показали цікавий факт, що зазвичай у більшості випадків люди, як правило, розміщують в останніх полях менше військ, ніж інших [16].

Більш відповідним цій роботі є експеримент, створений Модзелевським та іншими співавторами (2009), учасники якого відтворюють повторювану гру Блотто із п’ятдесятьма одиницями військ між п’ятьма полями (об’єктами) проти випадкового набору інших гравців. Автори цього дослідження виявили саме таку прихильність щодо перших чотирьох полів, як це було зроблено Чоудхурі та ін. Модзелєвський та співавтори показують, що ця схильність може бути пов'язана з тим, що їх люди, які проходили цей тест, є читачами системи "зліва на право". Це дослідження також показало, що багато з найбільш популярних стратегій були «великими трійками», де гравці вирішили зовсім не розміщувати на двох полях свої ресурси, щоб сконцентрувати ресурси на решті трьох областях в спробі виграти гру. Інший популярний вибір - рівновага (10, 10, 10, 10, 10).

Робота Арада-Рубінштейна 2010 року є цінною для студента обчислювальної економіки, оскільки це єдиний експериментальний документ, який пропонує точні результати турніру та результати для конкретних стратегій в круговому турнірі Блотто [17].

У 2008 році Арад і Рубінштейн провели турнір "Блотто", в якому кожен гравець мав розташувати 120 одиниць ресурсів між шістьма полями. Учасникам було надано інструкції, що містять правила гри та онлайн-форму, за допомогою якої можна було ввести обрану стратегію. Дві групи учасників представили стратегії гри: теорії ігор студентів з різних країн і читачів ізраїльського ділового журналу Calcalalyst. Як тільки стратегії були зібрані з усіх груп, турніри з турнірами з круглогодичного турніру були запущені з використанням наборів стратегій (кожна стратегія грала одна одну стратегію в наборі в грі Блотто рівно один раз), а також середні оцінки були наведені в таблицю. Стратегії потім класифікувалися середніми результатами турнірів. На таблиці 3.1 показані результати Арада-Рубінштейна (2010 р.) [16]:

Таблиця 3.1 – Результати експериментального турніру гри Блотто проведені Арадом та Рубінштейном у 2010р

| Classes' grand tournament | | | | | | | | Calcalist tournament | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | Mean Score | | 1 | 2 | 3 | 4 | 5 | 6 | Mean Score |
| 1 | **2** | **31** | **31** | **31** | **23** | **2** | 3.83 | 1 | **2** | **31** | **31** | **31** | **23** | **2** | 3.77 |
| 2 | 3 | 31 | 31 | 31 | 21 | 3 | 3.80 | 2 | 2 | 32 | 31 | 31 | 22 | 2 | 3.76 |
| 3 | 3 | 31 | 3 | 31 | 31 | 21 | 3.76 | 3 | 2 | 23 | 31 | 31 | 31 | 2 | 3.75 |
| 4 | **1** | **31** | **31** | **31** | **25** | **1** | 3.76 | 4 | 1 | 1 | 32 | 32 | 32 | 22 | 3.72 |
| 5 | **2** | **27** | **31** | **31** | **27** | **2** | 3.75 | 5 | **1** | **1** | **31** | **31** | **31** | **25** | 3.71 |
| 6 | 2 | 31 | 23 | 31 | 31 | 2 | 3.74 | 6 | **2** | **27** | **31** | **31** | **27** | **2** | 3.71 |
| 7 | **1** | **1** | **31** | **31** | **31** | **25** | 3.73 | 7 | 2 | 31 | 1 | 31 | 31 | 24 | 3.70 |
| 8 | 2 | 21 | 32 | 32 | 2 | 31 | 3.72 | 8 | **1** | **31** | **31** | **31** | **25** | **1** | 3.70 |
| 9 | 1 | 1 | 31 | 31 | 25 | 31 | 3.71 | 9 | 1 | 25 | 31 | 31 | 31 | 1 | 3.69 |
| 10 | 1 | 31 | 31 | 25 | 31 | 1 | 3.69 | 10 | 1 | 1 | 34 | 31 | 31 | 22 | 3.69 |

Таблиця 3.1 висвітлює дивовижний результат. У обох турнірах із круглого столу однаковий вектор стратегії (2, 31, 31, 31, 23, 2) мав найвищий середній бал, а підстановки цієї стратегії тримають інші місця у списку десятих. Чому саме ця стратегія (2, 31, 31, 31, 23, 2) була виграшною? Чому не було найбільш інтуїтивно зрозумілої стратегії у списку, такої як рівномірний розподіл на всі об’єкти - (20, 20, 20, 20, 20, 20)? Чи існувало щось спільне між найкращими стратегіями в кругових турнірах Блотто? Це були деякі питання, за якими сильна комп'ютерна імітаційна модель, основана на агентах, могла б дати деякі відповіді.

Було кілька спроб створити комп'ютерне моделювання гри Блотто. У своїх роботах 2007 року Тофіас використовує симуляцію для створення наборів найкращих стратегій для ігор Блотто, в яких існує асиметрія між кількістю інформації, доступною обом гравцям. Агент Тофіаса генерує стратегії за допомогою так званого "генетичного алгоритму" для створення стратегій, які найкращим чином відповідають стратегії, створеному його противником. Результати Тофіас показують, що, оскільки кількість полів в іграх Блотто збільшується, для агента з більшими ресурсами потрібно більше часу, щоб отримати перевагу у повторній грі. Проте Тофіас і пізніше робота Джексона (2010 року) обидва стикаються з визначенням відповідних параметрів для включення даної стратегії в набір "найкращих стратегій".

На мій погляд, це перший документ, який намагається перевірити сукупність експериментальних даних за допомогою методів симуляції. Це дивно, адже моделювання на основі агента, здається, є ідеальним партнером для експериментальної теорії ігор. Якщо ми зможемо правильно використовувати симуляційні методи для моделювання поведінки людей у ситуаціях з дуже чітко визначеними правилами та результатами, то, можливо, майбутні дослідники зможуть використовувати моделювання на основі агентів для розв'язання та моделювання більш складних людських взаємодій.

У порівнянні з іншими іграми з теорії ігор, гра Блотто є особливо сильним кандидатом на дослідження за допомогою симуляційного підходу.

Частково це пояснюється складністю моделювання гри дискретно і абсолютної кількості стратегій, можливих у грі Блотто. Навіть рішення для безперервної гри Блотто, як це вирішено Роберсоном (2006), дає мало користі, розглядаючи емпіричну або експериментальну ситуацію, таку як у результатах Арада-Рубінштейна (2010). Хоча в людей, можливо, виникають проблеми при аналізі багатьох тисяч стратегій, щоб знайти найкращі стратегії, це є простою роботою для комп'ютерного моделювання. На щастя, існує проста у вивченні, але потужна мова, яка робить цей тип моделювання порівняно легким. Як стверджує Ісаак (2008), гнучкі структури даних Python та відносна легкість кодування роблять цю мову, як добре підходящу для моделювання еволюційних ігор. Використовуючи структуру списку змінних Python, ми можемо легко відслідковувати стратегії, збирати найкращі стратегії та звітувати за результатами багатьох ігор гри Блотто, це є одними з декількох зручностей. Крім того, за допомогою фреймворку на основі агента для моделювання ми можемо спробувати імітувати людські показники в турнірі Блотто і відтворити експериментальні результати Арада та Рубінштейна застосовуючи комп'ютер.

Метою цього проекту є створення комп'ютеризованої моделі, яка зможе імітувати кругові турніри Блотто з використанням різних вхідних параметрів, щоб визначити, які стратегії найбільш успішні в різних іграх Блотто. Для цього ми повинні спочатку розробити спосіб генерації цілих векторів стратегії Блотто з нуля. Враховуючи неінтуитивну стратегію, яка була переможцем турніру Арада-Рубінштейна, не слід залишатись програмісту, щоб виробити стратегії для тестування, особливо в складних іграх. Створивши набір стратегій, порівняно просто провести круглий турбот Блотто. Турнір передбачає кожну стратегію у стратегії, яка відтворює стратегію у сеті, а також у грі Блотто. Звідти можна зібрати результати та проаналізувати моделі.

Проте, з кінцевою метою повторювання та пояснення стратегії виграшу Арада-Рубінштейна, просто виділення випадкових стратегічних векторів

недостатньо. Хоча люди не можуть проаналізувати кожну стратегію в грі Блотто, перш ніж вони прийдуть до вибору стратегії, експериментальні результати показують нам, що інтуїція про структуру гри зазвичай достатня, щоб негайно скинути деякі стратегії. Наприклад, небагато людей вибере стратегію, яка зосереджує всі ресурси в одному полі в багатопрофільній грі, оскільки це є втратою вогню. І все ж, коли комп'ютер створює випадкові стратегії, він не може використовувати ту саму інтуїцію, щоб відкидати погані стратегії та обирати хороші.

Отже, ще однією головною метою проекту є створення критеріїв відбору, які дозволять комп'ютеру відокремити хороші стратегії (тобто ті, які люди частіше грають) від поганих. Використовуючи комп'ютер для створення набору лише хороших стратегій, ми можемо більш точно імітувати набори стратегій, які виникають, коли люди грають в гру. Таким чином ми зможемо краще зрозуміти, чому стратегія виграшу Арад-Рубінштейна була переможцем у двох окремих турнірах Блотто.

## Висновки до розділу 3

У даному розділі було розглянуто кілька важливих етапів, що використовуються при дослідженні ГПБ, серед яких є проведення симуляції справжньої гри. Були розглянуті можливі кроки для генерація множин стратегій гри, проведення порівнянь стратегій (імітація поєдинку за об’єкт), створення рейтингу стратегій за декількома показниками та рекомендацій для знаходження кращих варіантів стратегій.

Були наведені результати декількох експериментальних турнірів, що відображають певні особливості, які потрібно враховувати при побудові методу знаходження кращих стратегій у грі полковника Блотто.

Такі дані можуть бути важливими при побудові методу знаходження оптимальних стратегій ГПБ.

# 4 МОДИФІКАЦІЯ АУКЦІОННОЇ МОДЕЛІ ГПБ

Серед моделей що були розглянуті вище видно що більш дослідженими випадками є симетричні ігри полковника Блотто, або ті у яких вектор цінностей об’єктів у двох опонентів співпадають.

Хоча у реальному житі ця ситуація є більше винятком, аніж чимось загальним. Для одного гравця певний об’єкт може мати дуже велике значення, а для іншого, наприклад, цей об’єкт майже ніякої цінності нести не буде.

У такому випадку перед гравцями постає проблема, яким саме чином розподіляти свої ресурси між об’єктами в залежності від різних оцінок важливості об’єктів між двома гравцями.

## 4.1 Рекомендації щодо побудови розподілу ресурсів гравців

Розглянемо алгоритм, що інструктує гравців, як потрібно послідовно розподіляти свої ресурси між об’єктами. На кожній ітерації алгоритму обидва гравці одночасно відправляють одиницю ресурсу на об’єкт, який вони найбільш цінують (серед об’єктів, які кожен гравець не вигравали).

Цей алгоритм досягає унікального профілю розгортання (unique deployment profile), коли гравці ніколи не залишаються байдужими при виборі об’єктів. Однак нам потрібно додати кілька уточнень для розгляду випадків байдужості (ці уточнення дозволять досягти профілю розгортання, який є унікальною рівновагою в чистих стратегіях, коли вона існує).

Щоб проілюструвати ці випадки, ми знову розглянемо ситуацію з трьома об’єктами і трьома одиницями ресурсів, де цінності перемог на об’єктах для гравців: $t_1=(7, 4, 1)$ *і* $t_2 = (2, 5, 5)$. Не маючи правила, алгоритм може досягти профілю розгортання *(1, 2, 0)* для одного з гравців і *(0, 1, 2)* для

іншого гравця - цей профіль не є рівновагою. Однак ми могли б також досягти профілю розгортання $(t_1, t_2) = ((1, 2, 0), (0, 2, 1))$, який дійсно є рівновагою в чистих стратегіях. Уточнення, яке дозволяє нам вибрати другий профіль розгортання, говорить наступне: щоразу, коли гравець досягає ітерації, в якій він байдужий до різних об'єктів, він відправляє свої ресурси на об'єкт, що є найменш цінним у противника (серед тих, до яких він байдужий).

Друге уточнення допомагає гравцю виділяти свої ресурси, коли перше уточнення все одно залишає його байдужим до різних об'єктів: коли немає жодного об'єкта, що є найменш переважним у його противника (серед тих, до яких гравець байдужий), гравець повинен відправляти свої ресурси на об'єкт, де були розгорнуті найменша кількість об'єктів. Це останнє уточнення дозволяє рівномірно розподіляти ресурси, коли гравці байдужі до багатьох об'єктів і дозволяє їм досягти унікального рівноваги, коли обидва гравці однаково оцінюють все об'єкти.

Попередній алгоритм (разом з двома уточненнями) дозволяє гравцям розподіляти всі свої ресурси. Більш того, остаточне розподілення ресурсів однозначно визначено (за винятком деяких випадків, коли на об'єктах є байдужість). Найцікавіше, що в наступному твердженні йдеться, що всякий раз, коли існує рівновага у чистих стратегіях, наведений вище алгоритм досягає такого розподілу.

Розглянемо ще раз той самий випадок, коли в нас є три об'єкти з трьома одиницями ресурсів в кожного гравця, де цінності перемог на об'єктах для гравців: $t_1=(7, 4, 1)$ *і* $t_2 = (2, 5, 5)$. У цьому випадку найкращими стратегіями розподілу ресурсів є: $((1, 2, 0), (0, 2, 1))$. Але нехай другий гравець буде вважати що вектор цінностей першого гравця є не $t_1= (7, 4, 1)$, а $t_1= (7, 1, 4)$. То для цього випадку другому гравцю було б доречніше розташувати свої ресурси наступним чином - $(0, 1, 2)$. В результаті чого замість початкової нічиї, другий гравець досягне поразки.

Розглянемо це більш детально. Гравці розташовують основні свої ресурси на об’єктах, які є більш цінними для них, та об’єктах, що можливо є менш цінними але потрібними для здобування перемоги у всій грі. І ті об’єкти на яких більш суперечлива ситуація, на них і є сенс додавати трохи більше ресурсів, ніж на цінніші об’єкти.

Цей підхід і змусив розподілити більшість ресурсів на другому об’єкті, де він знаходиться у суперечливій ситуації. Третій об’єкт найменш привабливий для першого гравця, тому навряд чи він туди відправить більшість ресурсів. Перший об’єкт для першого гравця найцінніший, то з дуже великою ймовірністю туди будуть розподілені ресурси першого гравця. Для першого другий об’єкт теж цінний, тому туди і може бути направлена теж частина ресурсів. Згідно з цим другий об’єкт для другого гравця є суперечливим. І тому він вважає за доречне – відправити більшість своїх ресурсів.

Але при помилковій оцінці цінностей (не $t_1 = (7, 4, 1)$, а $t_1 = (7, 1, 4)$), другий гравець розподіляє свої ресурси не на той об’єкт, що призводить до поразки у грі. Як видно із прикладу, нав’язування неправдоподібних оцінок намірів/бажань/векторів цінностей гравців може сильно допомогти у ході гри. Методи рефлексивного управління можуть дати такі результати [19].

Також є можливість розвинути модель ГПБ із різними векторами цінностей у гравців, шляхом додавання рефлексивної складової під час формування розподілу ресурсів на об’єкти.

Таким чином якщо перший гравець має вектор розподілу *(1, 2, 0)*, а другий - *(0, 2, 1)*. Перший може обміркувати/передбачити хід думок другого гравця та розташувати свої ресурси наступним чином - *(1, 0, 2)*. Перший гравець можливо і отримає меншу вигоду від можливих перемог на об’єктах, але з більшою ймовірністю йому вдасться виграти усю гру, заволодівши більшістю об’єктів.

## 4.2 Модифікована модель

Як це було показано раніше, будь-яку ГПБ можна розписати через побудову матриць виграшів гравців. Та потім серед них знайти ті стратегії, що будуть дуже близькими до оптимальної, дивлячись на кількість перемог під час попарних порівнянь усіх стратегій або на ті, що потрапили у множину стратегій рівноваги Неша.

Побудова таких матриць є дуже простим та майже стовідсотковим рішенням, але цей метод буде добре працювати лише для малої кількості об’єктів (в межах п’яти) та одиниць ресурсів, які розподіляє гравець (менше 50). Проблема виникає в тому, що розмірність матриці росте у факторіальній залежності від кількості об’єктів. А при дуже великих матрицях, по-перше її треба десь зберігати, а по-друге на обробку такого об’єму даних може піти дуже багато ресурсів комп’ютерних потужностей.

Через це виникає потреба знайти інший спосіб знаходження оптимальних стратегій для гравців, при умовах що:

- гравці можуть мати різну кількість ресурсів,
- вектори корисності від перемоги на об’єкті можу відрізнятися,
- кількість об’єктів може бути більша за 5, 10 і т.д.,
- кількість одиниць ресурсів у гравців можуть бути більші за 50,100 і т.д.

Розглянемо функцію виплат гравців, що лежить у основі побудови матриць виплат:

$$f_x(x,y) = \sum_{\text{i}=1}^{n} f_{i\,x}(x_i, y_i) = \sum_{\text{i}=1}^{n} X_{\text{i}} * I(x_i > y_i) + \frac{1}{2}\sum_{\text{i}=1}^{n} X_{\text{i}} * I(x_i = y_i)$$

Уявімо що ми маємо певний вектор розподілу ресурсів суперника $y$.

То для нього є можливість переглянути усі можливі стратегії від першого гравця у відповідь на дії суперника $y$. Але тут ми знову можемо зіткнутися з проблемою дуже великої матриці виплат.

В якості виходу з цієї ситуації можна розглядати не всі підряд стратегії першого гравця, а провести розбиття стратегій по три класи для кожного об'єкту. Для роз'яснення можна провести розбиття для гри Блотто, де буде тільки один об'єкт за який будуть конкурувати обидва гравці. Нехай суперник розмістить на об'єкті $y_1$ ресурсів. Тоді множину усіх стратегій першого гравця є можливість розподілити на три категорії (класи) наступним чином:

- В перший клас будуть входити усі стратегії першого гравця, в яких він відправить на об'єкт своїх ресурсів менше, аніж це зробив другий гравець.
- В другому класі будуть знаходитися усі стратегії першого гравця, в яких він відправить на об'єкт своїх ресурсів стільки ж, на скільки це зробив і другий гравець.
- А в третьому класі будуть знаходитися усі стратегії першого гравця, в яких він відправить на об'єкт своїх ресурсів більше, ніж це зробив другий гравець.

За таким розподілом на класи можна пройтися по кожному з об'єктів (якщо їх декілька), та виходячи з цього тепер в нас буде можливість розбити функцію виграшу на три складові в залежності від стану системи.

Наприклад, якщо у грі є два об'єкти, кількості ресурсів обох гравців дорівнюють п'яти ($R_x = R_y = 5$), цінності перемоги на об'єктах для першого гравця - $X_1 = 5, X_2 = 4$, а для другого - $Y_1 = 3,\ Y_2 = 4$. Та нехай ми ще й знаємо (або припускаємо), що вибір розташування ресурсів другого гравця буде дорівнювати $y = (2, 3)$. То виходячи з цих даних, для рішення нам не обов'язково розглядати усі можливі комбінації розташування першого гравця, замість цього можна виконати розбиття стратегій на класи. Для спрощення пояснення введемо вектор $S$, розмірність якого буде дорівнювати кількості об'єктів задачі, а значеннями у $i$-их комірках $s_i$ будуть наступні

символи «<», «=», «>». Знак «<» буде вказувати на те, що перший гравець виділить ресурсів на об'єкт з індексом де опинився цей знак ($s_i$) менше за другого гравця. Знак «>» буде вказувати на те, що перший гравець виділить ресурсів на $s_i$ більше за другого гравця. А знак «=» - що кількість ресурсів що планують виділити обидва гравці на певний *i*-ий об'єкт ($s_i$) буде рівною. Такі позначення допомагають визначити до якого класу потрібно віднести стратегію. Виходячи з цього повний перебір стратегій першого гравця можна виконати наступним чином:

- Випадок де вектор *S* має такі значення (<, <), нам не підходить, бо не всі ресурси використовуються гравцем. Монотонність функції виграшу (зростаюча) вказує на те, що при неповному розподілі ресурсів є ризик недоотримати певну частину можливого виграшу;

- Випадок *S=(*<, *=)* не підходить, бо не всі ресурси використовуються гравцем;

- Випадок *S=(*<, >*)*: гравець не буде намагатися перемогти на першому об'єкті та буде зосереджувати свої сили (ресурси) на другому. Тоді очікуваним виграшем гравця буде $U_A$=0+4=4. Серед можливих варіантів вектору розташування ресурсів є приклад x = (1, 4);

- Випадки *S*=(=,<) та *S*=(=,>): ∅ (порожня множина). Через те що перший випадок вказує на неповне використання ресурсів, а другий – використання ресурсів, яких гравець не має, що в даний момент неможливо;

- Випадок *S*=(=, =): гравець планує розмістити свої ресурси так само, як і суперник. В результаті цього розподілу його очікуваний виграш буде дорівнювати $U_A$=5/2+4/2 =4,5. Приклад можливого розташування: *x* = (2, 3);

- Випадок *S*=(>, <): гравець відмовляється від перемоги на другому об'єкті, та зосереджує свої сили (ресурси) на першому. $U_A$=5+0=5. Приклад можливого розташування: *x* = (4, 1);

- Випадки $S$=(>,=) та $S$=(>,>): ∅. Через те що перший і другий випадок вказують на використання ресурсів, яких гравець не має, що в даний момент неможливо.

Більшість із цим розбиттів, як ми бачимо, можна одразу відкидати через неможливість знаходження таких підмножин стратегій, або їх неефективність.

Аналізуючи результати ми можемо побачити що найкращим для першого гравця (при умові що $y$ = (2, 3) ) виявився випадок $S$= (>, <). Тобто, якщо перший гравець розмістить своїх ресурсів на перший об'єкт більше за кількість другого (що дорівнює 2), та на другий об'єкт менше за другого (3), то перший гравець отримає виграш у розмірі $U_A$=5.

Очевидно що цей метод суттєво скорочує множину можливих варіантів розташування. Але в цьому рішенні є декілька нюансів:

- метод не надає точних рішень, якийсь один вектор розташування ресурсів, а лише діапазони у яких можна знайти шуканий вектор;
- метод залежить від того, яким чином суперник розподілить свої сили/ресурси (вектор початкового розподілу).

Щодо побудови вектору точного розташування ресурсів, можна запропонувати декілька варіантів рішення цього питання:

- у випадках коли стратегія потрапляє до класу в якому на одному з об'єктів потрібно розташувати ресурсів менше ніж суперник – ми можемо нічого не розміщувати на тому об'єкті;
- у випадках коли вектор $S$ має в собі декілька значень «>», тобто гравцю на таких об'єктах потрібно розташувати свої ресурсів більше аніж це зробить суперник, може постати питання, а яким саме чином краще та ефективніше розподілити свої ресурси. В цьому розділі буде винесено два способи, що на мій погляд є кращими рішеннями у цій ситуації. Першою є розміщувати залишок, що є результатом дії різниці між усіма ресурсами та тими що будуть розподіленні на об'єкти де ми виділяємо ресурсів менше або стільки ж як і суперник, порівну між об'єктами, для яких відповідна координата

вектору $S$ має знак «>» або розподілити залишок на таких об'єктах пропорційно до відносних вигод об'єктів (як їх обчислювати буде описано нижче).

У існуючих відкритих джерелах інформації я не знайшов методу, за допомогою якого можна чітко виявити важливість або корисність від кожного об'єкта для ігор Блотто у яких гравці можуть мати різні вектори корисностей об'єктів. Тож я вирішив запропонувати свій варіант розстановки пріоритетів вибору куди краще розмістити більше ресурсів, а куди менше.

У випадку коли вектор цінностей об'єктів $(\forall\, i \in [1,n]: X_i = Y_i)$ для гравців є спільним, вектор пріоритетів мав наступний вигляд:

$Pr = (\frac{V_1}{V},.., \frac{V_n}{V})$, де $V_i$ $(i \in [1,n], n \in N)$ - виграш, який може бути отриманий при перевазі на $i$ об'єкті одним з гравців, а V = $\sum_{i=1}^{n} V_i$ – сума по $i$.

Розпишемо відносні пріоритети для кожного з гравців де вектори цінностей можуть відрізнятися:

$Pr_A = (\frac{X_1}{V_A},.., \frac{X_n}{V_A})$, де $V_A = \sum_{i=1}^{n} X_i$ – сумарний виграш для гравця А,

$Pr_B = (\frac{Y_1}{V_B},.., \frac{Y_n}{V_B})$, де $V_B = \sum_{i=1}^{n} Y_i$ – сумарний виграш для гравця В.

Для тих об'єктів які є більш пріоритетними для суперника є сенс розміщувати свої ресурси в меншій кількості, а ніж на тих об'єктах які є корисні для нас та не мають великої цінності для суперника. Тому було запропоновано обчислювати відносні пріоритетності для гравців наступним чином:

$$Pr_{A(\text{B})\,i} = \frac{\frac{X_i}{V_A}*(1-\frac{Y_i}{V_B})}{\sum_{i=1}^{n}\frac{X_i}{V_A}*(1-\frac{Y_i}{V_B})} \qquad Pr_{B(A)\,i} = \frac{\frac{Y_i}{V_B}*(1-\frac{X_i}{V_A})}{\sum_{i=1}^{n}\frac{Y_i}{V_B}*(1-\frac{X_i}{V_A})}$$

В результаті ми отримаємо вектори корисностей, що враховують різні вектори виграшів, що можуть отримати гравці у разі перемоги на об'єктах.

Повернемося до самого відбору стратегій.

Використовуючи вище описане розбиття множини усіх стратегій гравця на класи, ми можемо з кожного класу взяти лише кращі елементи,

через те що гірші будуть відкинуті як доміновані. Таким чином ми отримаємо підмножину стратегій, що буде доцільно розглядати далі.

Єдиною особливістю цього підходу є розбиття усіх стратегій гравця на класи відштовхуючись від значень вектору розподілу ресурсів суперника, який на початку ми самі генеруємо. А такий підхід може дати помилкові дані.

Виходом з цієї ситуації можна використати такий підхід, але не для гравця, а його суперника, використовуючи перший результат як вектор розподілу ресурсів першим гравцем. Таким чином ми можливо покращимо той вектор розподілу ресурсів другого гравця, який ми штучно генерували.

Після цього є можливість повторити дослідження для першого гравця, відштовхуючись від ново отриманого вектору розподілу.

Тобто для отримання покращених результатів програми, ми можемо ітеративно використати такі методи, для яких будуть використані ново отримані результати. Це надасть можливість кожного разу потенційно отримати кращі множини альтернатив. Зібравши які у одну спільну множину, ми матимемо множину альтернатив, які далі вже можна дослідити використовуючи інші методи.

Серед методів що можна використати далі був вибраний метод біматричної гри.

На основі отриманих множин, підмножини яких було отримано вище описаними методами при певних умовах – різних векторах розподілу ресурсів першого та другого гравця, ми будуємо дві матриці виграшів.

Виграш для першого гравця у координаті $(i, j)$ матриці $A$ обчислюється шляхом порівняння $i$-ої стратегії першого гравця та $j$-ої стратегії другого гравця. Відбувається пооб’єктне порівняння кількості ресурсів, що були відправлені гравцями. Якщо перший гравець відправив більше ресурсів за другого на певний об’єкт, то за це він отримає свій виграш за той об’єкт у повній мірі. У випадку рівної кількості ресурсів, перший гравець отримає половину виграшу. Інакше – виграшу не буде.

Таким самим чином відбувається побудова матриці *B* виграшів для другого гравця, зі збереженням порядку гравців під час порівняння стратегій. Отримавши дві матриці виграшів перейдемо до самого знаходження рішення біматричної гри.

Для кожного стовпця матриці *A* знайдемо максимальний елемент. Положення таких елементів відповідає прийнятним ситуаціям першого гравця, коли супротивник обрав стратегію, що відповідає тому стовпцю.

Для кожного рядку матриці виграшів *B* виберемо найбільший елемент. Положення таких елементів буде визначати прийнятні ситуації другого гравця, коли перший обрав стратегію, що відповідає тому рядку.

Далі потрібно скласти дві множини для двох гравців на основі двох матриць *A* та *B*, шляхом виписування координат таких максимумів у відповідних таблицях, де перша координата є номером рядка максимального елемента, а друга – стовпця.

І тепер якщо ми перетнемо ці множини, то отримаємо підмножину з елементів, що буде рівновагами по Нешу. Такі ситуації виявилися оптимальними за Парето для обох гравців.

На прикладі одноелементної такої множини ми можемо отримати номер стратегії що є оптимальною для вибору першим гравцем, та другим. Її номером є перша координата цього елемента, або друга – для другого гравця.

## 4.3 Дослідження запропонованої моделі

Наведемо приклади досліджень моделі на основі повного аналізу множини можливих стратегій та скорочену множену, знаходження якої було запропоновано у даній роботі.

Розглянемо ГПБ, де кількість об’єктів буде дорівнювати трьом, кількість ресурсів в кожного з двох гравців дорівнює трьом. А ось вектори корисності об’єктів для гравців будуть відрізнятися. Для першого цей вектор

буде дорівнювати $t_1=(7, 4, 1)$, а для другого - $t_2 = (2, 5, 5)$, як це було описано у прикладі з пункту 4.1.

Побудуємо платіжну матрицю для гравця *A*, шляхом пооб'єктного порівняння відповідних стратегій першого та другого гравця. Сама матриця продемонстрована у таблиці 4.1 та знаходиться у рамці. Зліва від матриці були розташовані відповідні стратегії першого гравця *A*, а згори – стратегії гравця *B*.

У наступному рядку знизу після матриці записані значення, що є максимальними у відповідних стовпцях. А два останні рядки містять в собі координати оптимальних пар для першого гравця, де першою координатою $X_i$ є номери стратегій що мають максимальне значення у порівнянні з *j*-ою ($Y_j$) стратегією суперника *B*.

Таблиця 4.1 - Платіжна матриця для гравця *A*.

| | | | 3 | 0 | 0 | 2 | 2 | 1 | 1 | 0 | 0 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | 0 | 3 | 0 | 1 | 0 | 2 | 0 | 2 | 1 | 1 |
| | | 1(A)\2(B) | 0 | 0 | 3 | 0 | 1 | 0 | 2 | 1 | 2 | 1 |
| 3 | 0 | 0 | 3.5 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 |
| 0 | 3 | 0 | 4 | 2 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| 0 | 0 | 3 | 2 | 2 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 2 | 1 | 0 | 4 | 7 | 11 | 5.5 | 7.5 | 7 | 11 | 7 | 9 | 9 |
| 2 | 0 | 1 | 2 | 9 | 7 | 5.5 | 4.5 | 9 | 7 | 8 | 7 | 8 |
| 1 | 2 | 0 | 4 | 7 | 11 | 4 | 4 | 5.5 | 7.5 | 9 | 11 | 7.5 |
| 1 | 0 | 2 | 2 | 9 | 7 | 2 | 2 | 5.5 | 4.5 | 9 | 8 | 5.5 |
| 0 | 2 | 1 | 6 | 2 | 4 | 6 | 5 | 4 | 4 | 3 | 4 | 5 |
| 0 | 1 | 2 | 6 | 2 | 4 | 4 | 6 | 2 | 5 | 2 | 3 | 4 |
| 1 | 1 | 1 | 6 | 9 | 11 | 4 | 5 | 5.5 | 7.5 | 8 | 9 | 6.5 |
| | | max | 6 | 9 | 11 | 7 | 7.5 | 9 | 11 | 9 | 11 | 9 |
| | стр | X | 8.9.10. | 5.7.10. | 4.6.10. | 1 | 4 | 5 | 4 | 6.7 | 6 | 4 |
| | | Y | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

Таким чином ми можемо отримати наступну множину пар стратегій, що є позиціями максимумів в стовпцях матриці *A*: (8,1), (9,1), (10,1), (5,2), (7,2), (10,2), (4,3), (6,3), (10,3), (1,4), (4,5), (5,6), (4,7), (6,8), (7,8), (6,9), (4,10).

Таку ж саму процедуру треба виконати і зі сторони гравця *B*, для побудови його матриці виграшу – таблиця 4.2. В цьому випадку збоку самої матриці представлені можливі стратегії гравця *B*, а згори – гравця *A*. Так само знаходимо максимальні значення у стовпцях, та записуємо у останніх двох рядках координати таких значень.

Таблиця 4.2 - Платіжна матриця для гравця *B* (з боку *B*).

| | | | 3 | 0 | 0 | 2 | 2 | 1 | 1 | 0 | 0 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | 0 | 3 | 0 | 1 | 0 | 2 | 0 | 2 | 1 | 1 |
| | | 2(B)\1(A) | 0 | 0 | 3 | 0 | 1 | 0 | 2 | 1 | 2 | 1 |
| 3 | 0 | 0 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 0 | 3 | 0 | 5 | 2.5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| 0 | 0 | 3 | 5 | 5 | 2.5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| 2 | 1 | 0 | 5 | 2 | 7 | 3.5 | 6 | 2 | 7 | 2 | 4.5 | 4.5 |
| 2 | 0 | 1 | 5 | 7 | 2 | 6 | 3.5 | 7 | 2 | 4.5 | 2 | 4.5 |
| 1 | 2 | 0 | 5 | 2 | 7 | 5 | 5 | 3.5 | 6 | 4.5 | 7 | 6 |
| 1 | 0 | 2 | 5 | 7 | 2 | 5 | 5 | 6 | 3.5 | 7 | 4.5 | 6 |
| 0 | 2 | 1 | 10 | 5 | 5 | 10 | 7.5 | 7.5 | 5 | 5 | 5 | 7.5 |
| 0 | 1 | 2 | 10 | 5 | 5 | 7.5 | 10 | 5 | 7.5 | 5 | 5 | 7.5 |
| 1 | 1 | 1 | 10 | 7 | 7 | 7.5 | 7.5 | 6 | 6 | 4.5 | 4.5 | 6 |
| | | max | 10 | 7 | 7 | 10 | 10 | 7.5 | 7.5 | 7 | 7 | 7.5 |
| | стр | Y | 8.9.10 | 5.7.10. | 4.6.10. | 8 | 9 | 8 | 9 | 7 | 6 | 8.9. |
| | | X | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

З цієї матриці отримаємо теж свою множину пар стратегій, що є позиціями максимумів в стовпцях матриці *B*: (1,8), (1,9), (1,10), (2,5), (2,7), (2,10), (3,4), (3,6), (3,10), (4,8), (5,9), (6,8), (7,9), (8,7), (9,6), (10,8), (10,9).

Результатом перетину цих двох множин є множина з однієї пари: (6;8).

Таким чином, була знайдена одна рівноважна ситуація Неша (6; 8). Ця ситуація опинилися оптимальною по Парето для обох гравців.

Отримана пара (6,8) дає нам роз’яснення з приводу, що оптимальною стратегією для першого гравця вважається шоста, а саме (1, 2, 0). Та для другого гравця є восьма, а саме – (0, 2, 1).

У такій рівноважній ситуації перший гравець *A* отримає виграш у розмірі 9 одиниць, а другий гравець *B* - 7.5 одиниць.

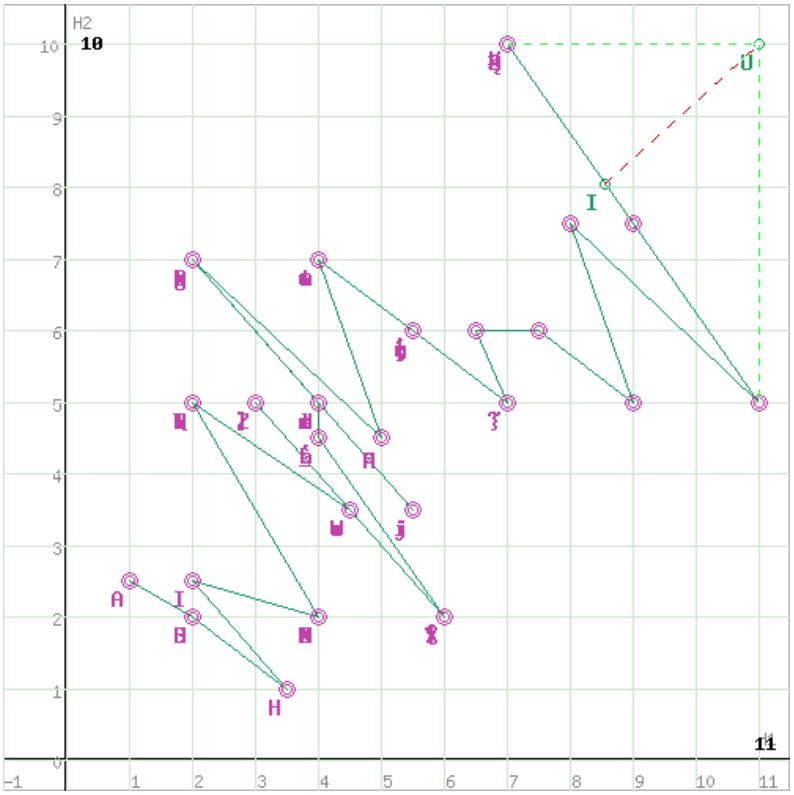


Рисунок 4.1 - Множина всіх векторів виграшів, що можливо реалізувати для даної гри

Як бачимо з рисунку 4.1, гра що розглядається має декілька ситуацій що є оптимальними по Парето, але ситуація (пара стратегій) з найбільшим виграшем, яку можна знайти за методом ідеальної точки є *I*(8.561; 8.049).

А перед цю ж саму задачу буде розглянуто з використанням методів зменшення кількості елементів у множинах зі стратегіями, що є доцільно розглядати у задачі.

Як було описано вище, одним з перших кроків потрібно визначити початковий вектор розподілу ресурсів суперником поміж усіма об’єктами. Нехай цей вектор буде дорівнювати $y$ = (1, 1, 1). Тобто розподілимо його ресурси між об’єктами рівномірно.

Тепер є можливість знайти/підібрати найкращі стратегій для першого гравця. Під час пошуку ми будемо вирішувати задачу пакування рюкзака, намагатимемося максимізувати виграш відштовхуючись від векторів корисностей об’єктів, відносні пріоритети – вектори *Pr*. Залишок ресурсів розподіляється за відносними пріоритетами об’єктів. Результатом такого кроку буде множина найкращих стратегій, вибір яких дасть максимальний виграш при умові що ми знаємо вектор розподілу ресурсів опонента. У найкращому випадку ця множина буде складатися з одного об’єкта.

Далі за для наступного кроку ітерації потрібно так само використати хід із задачі пакування рюкзака, та знайти кращі відповіді на такий хід зі сторони опонента. Далі метод виконує ті ж самі операції ітеративно декілька разів. Серед запропонованих варіантів кількості ітерацій є кількість об’єктів у грі.

На кожній з таких ітерації ми отримаємо множину, при певних умовах, кращих стратегій. Їх ми додаємо до однієї глобальної множини. Пройшовши усі ітерації ми отримаємо дві такі глобальні множини найкращих стратегій, одну для першого гравця, а іншу для його опонента.

Приклад роботи програмного додатку на декількох ітераціях:

Iteration #0:

Allocation of sec player: 1 1 1

InequalitySignsForBestAllocation: > = <

BestPrice = 9

Best allocation of fir player: 2 1 0

Allocation of first player: 2 1 0

InequalitySignsForBestAllocation: < > >

BestPrice = 10

Best allocation of second player: 0 2 1

Iteration #1:

Allocation of sec player: 0 2 1

InequalitySignsForBestAllocation: > < >

BestPrice = 9

Best allocation of fir player: 1 0 2

Allocation of first player: 1 0 2

InequalitySignsForBestAllocation: < > =

BestPrice = 7.5

Best allocation of second player: 0 1 2

Iteration #2:

Allocation of sec player: 0 1 2

InequalitySignsForBestAllocation: > > <

BestPrice = 11

Best allocation of fir player: 1 2 0

Allocation of first player: 1 2 0

InequalitySignsForBestAllocation: < = >

BestPrice = 7.5

Best allocation of second player: 0 2 1

Після цього ми вже можемо будувати матричну гру, на основі двох множин стратегій гравців.

Серед найкращих стратегій першого гравця було обрано наступні: (2, 1, 0), (1, 0, 2), та (1, 2, 0).

А кращими стратегіями другого гравця було обрано такі: (1, 1, 1), (0, 2, 1), та (0, 1, 2).

Таблиця 4.3 - Платіжна матриця для гравця *A*.

| | | |
|---|---|---|
| **9** | 7 | 9 |
| 5.5 | **9** | 8 |
| 7.5 | **9** | **11** |

Позиції максимумів у стовпцях матриці *A* мають наступні координати: (1,1), (2,2), (3,2), (3,3).

Таблиця 4.4 - Платіжна матриця для гравця *B* (з боку *B*).

| | | |
|---|---|---|
| 7.5 | **10** | 7.5 |
| 6 | 5 | **7.5** |
| 6 | **7.5** | 5 |

Позиції максимумів у стовпцях матриці *B* мають наступні координати: (1,2), (2,3), (3,2).

Перетином таких двох множин є елемент (3, 2). Він є рівноважною ситуацією, та оптимальною за Парето для обох гравців.

Для цієї ситуації, як вже було описано у результатах попереднього методу, перший гравець отримає 9 одиниць виграшу, а другий – 7.5.

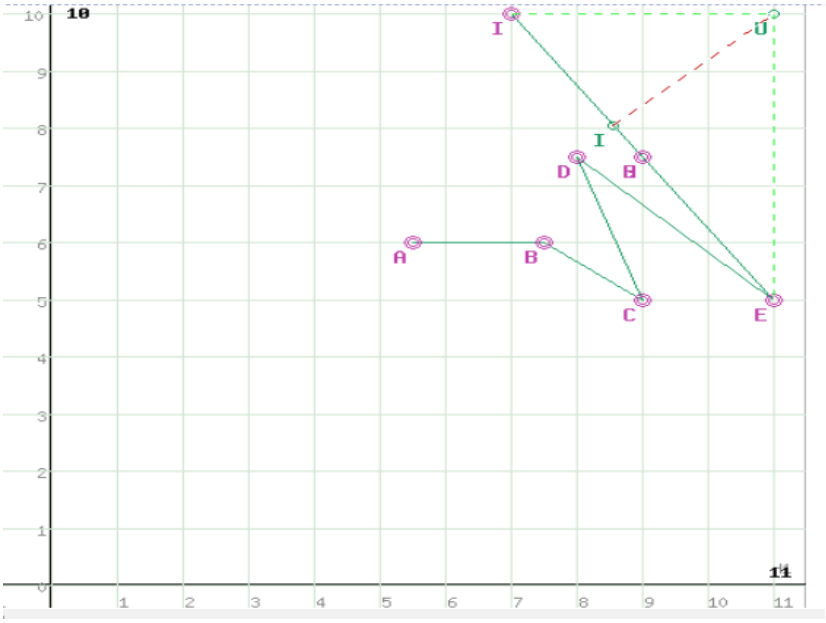


Рисунок 4.2 - Множина всіх векторів виграшів, що можливо реалізувати (метод 2)

Цей метод теж вказує на те, що для першого гравця найкращим варіантом буде обрати стратегію (1, 2, 0), а для другого гравця – (0, 2, 1). Такий вибір їм принесе по 9 та 7,5 одиниць виграшу.

Таким чином було отримано той самий результат що був описаний у методі, де будували біматричну гру на основі усіх можливих стратегій гравців, та той результат що був описаний ще у прикладі з пункту 4.1 з переліком рекомендацій побудови векторів розподілу ресурсів.

Також було розглянуто гру Блотто при умовах, що були описані у підрозділі 3.2. В якому є 6 об’єктів та 120 одиниць ресурсів в кожного з гравців. Усі об’єкти для гравців у цій задачі є рівноцінними.

Результатом роботи програмного додатку на таких умовах стали наступні, запропоновані для вибору першому гравцю, стратегії:

20 0 20 31 30 19 Victories - 602; Draws - 201; Losses - 96

12 0 23 32 31 22 Victories - 602; Draws - 167; Losses - 130

22 0 22 33 22 21 Victories - 600; Draws - 166; Losses - 133

20 0 20 31 19 30 Victories - 599; Draws - 203; Losses - 97

20 0 19 30 32 19 Victories - 598; Draws - 193; Losses - 108

20 0 19 19 32 30 Victories - 596; Draws - 204; Losses - 99

20 0 19 19 30 32 Victories - 595; Draws - 207; Losses - 97

0 20 20 20 30 30 Victories - 594; Draws - 230; Losses - 75

20 0 19 30 19 32 Victories - 594; Draws - 200; Losses - 105

20 0 19 31 31 19 Victories - 592; Draws - 205; Losses – 102

Наведені лише перші 10 стратегій з відсортованої, за кількістю попарних перемог, множини. В даному випадку значення Victories означає кількість стратегій, за які ця була краще за інших. Draws - кількість стратегій, що були рівноцінні, а Losses – ті що були гірші.

Згідно з результатами наведеними у підрозділі 3.2 спостерігається певна схожість у числах які були використані в якості координат векторів розподілу. Тільки не завжди їх місця співпадали у векторах.

Це вказую на те, що стаття розглядала данні реальних людей, поведінку яких не завжди легко спрогнозувати. Тим більш у випадках вибору на які об’єкти розподілити ресурси, якщо вони є рівноцінними.

Така проблема вирішується якраз шляхом встановлення пріоритетів об’єктів. Одним із очевидних рішень є встановлення значень корисності перемог на об’єктах. Такий механізм простіше реалізувати програмно. Що і було виконано у програмному додатку.

**Висновки до розділу 4**

У цьому розділі було запропоновано новий метод пошуку рішення для аукціонних моделей гри полковника Блотто. В тому числі був описаний алгоритм пошуку множини оптимальних стратегій розташування ресурсів гравцем, наведені рекомендації для побудови векторів розподілу ресурсів та принципи роботи із асиметричною моделлю, де гравці можуть мати різні вектори корисностей перемог на об’єктах у грі.

Результатом роботи програмного додатку, що описаний у додатку А є список найкращих стратегій, відсортований за кількістю стратегій які вони домінують з тієї множини у попарних порівняннях. При цьому максимальний виграш може бути досягнений декількома різними стратегіями. Для таких випадків може бути обрана будь-яка з них, через те що для кожної з них ймовірність отримати виграш буде коливатися в межах допустимого відхилення.

Також було продемонстровано результати роботи програмного додатку, при тих самих умовах що були описані для одного з прикладів ГПБ у підрозділі 4.1. Тобто при однакових початкових умовах гри було проведено порівняння результатів використання методу повного перебору стратегій та методу, що був запропонований у даній роботі (підрозділ 4.2). Результати роботи двох методів співпали із теоретичним результатом підрозділу 4.1. Що вказує на ефективність використання даного методу для пошуку оптимальних стратегій у ГПБ.

# ВИСНОВКИ

Проведений огляд існуючих моделей ГПБ вказує на актуальність досліджень цього класу ігрових моделей, з огляду на їх додатки для військових ігор, для розміщення ресурсів в іграх голосування, для пошуку оптимальних стратегій в одномоментних конкурсах/аукціонах, а також в силу відсутності повного аналітичного рішення в складних випадках.

У цій роботі були наведені приклади існуючих моделей ГПБ та їх дослідження. Описано результати проведених експериментів та кроки для проведення симуляція ГПБ.

В результаті оцінки переваг та недоліків існуючих методів, що пропонують варіанти розподілу ресурсів між об’єктами, був запропонований свій метод. Він працює у рамках аукціонної моделі ГПБ з певними модифікаціями. Для реалізації яких були застосовані концепції рефлексивних ігор, запропоновано новий спосіб встановлення пріоритетів вибору об’єктів та спосіб який значно скорочує кількість стратегій, які доцільно розглядати для пошуку оптимальних стратегій.

В роботі також було досліджено інформаційне управління в моделі інформаційної рефлексії гравців. У цій моделі гравці можуть мати хибні уявлення про параметри моделей  опонента. А це на свою чергу може призвести до покращення позицій гравця, опонент якого став цілю інформаційного управління.

Проведений аналіз результатів роботи програмного додатку на основі запропонованого методу підтверджує його коректність та ефективність.

Подальший розвиток методу може бути спрямований на покращення критеріїв за якими можна відсортувати об’єкти за їх важливістю. Потреба в цьому виникає в першу чергу для ситуацій коли об’єкти є рівно важливими, але люди можуть обрати той чи інший об’єкт в залежності від їх порядку або ще якогось чиннику.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ


1 Strategy_(game_theory) [Електронний ресурс] URL: https://en.wikipedia.org/wiki/Strategy_(game_theory) (дата звернення: 15.09.2018)

2 Матричні ігри [Електронний ресурс] URL: http://um.co.ua/2/2-7/2-75351.html (дата звернення: 15.09.2018)

3 Nash equilibrium [Електронний ресурс] URL: https://en.wikipedia.org/wiki/Nash_equilibrium (дата звернення: 15.09.2018)

4 Prisoner’s dilemma [Електронний ресурс] URL: https://en.wikipedia.org/wiki/Prisoner%27s_dilemma (дата звернення: 11.10.2018)

5 Lagrange multiplier [Електронний ресурс] URL: https://studfiles.net/preview/2398487/page:17/(дата звернення: 11.10.2018)

6 Нахождение количества разбиений числа на слагаемые [Електронний ресурс] URL: https://neerc.ifmo.ru/wiki/index.php?title=%D0%9D%D0%B0%D1%85%D0%BE%D0%B6%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BA%D0%BE%D0%BB%D0%B8%D1%87%D0%B5%D1%81%D1%82%D0%B2%D0%B0_%D1%80%D0%B0%D0%B7%D0%B1%D0%B8%D0%B5%D0%BD%D0%B8%D0%B9_%D1%87%D0%B8%D1%81%D0%BB%D0%B0_%D0%BD%D0%B0_%D1%81%D0%BB%D0%B0%D0%B3%D0%B0%D0%B5%D0%BC%D1%8B%D0%B5 (дата звернення: 06.11.2018)

7 Dynamic programming [Електронний ресурс] URL: https://en.wikipedia.org/wiki/Dynamic_programming (дата звернення: 13.11.2018)

8 Вентцель Е.С. Элементы теории игр. – М.: Физматгиз, 1961. – 68 с.

9 Borel E. La théorie du jeu les équations intégrales á noyau symétrique// Comptes Rendus de l'Académie. 1921. Vol. 173. P. 1304 – 1308.

10 Gross O., Wagner R. A Continuous Colonel Blotto Game / RAND Corporation RM-408, 1950.

11 Borel E., Ville J. Application de la théorie des probabilitiés aux jeux de hasard. – Paris: Gauthier-Villars, 1938. P. 105 – 113.

12 Roberson B. The Colonel Blotto Game // Economic Theory. 2006. Vol. 29. P. 1 – 24.

13 Hortala-Vallve R., Llorente-Saguer A. Pure strategy Nash equilibria in non-zero sum colonel Blotto games // International Journal of Game Theory. 2011.

14 Robson R.W. Multi-Item Contest / Australian National University. Working Paper № 446, 2005. – 27 p.

15 Новиков Д.А., Чхартишвили А.Г. Рефлексивные игры. – М.: Синтег, 2003. – 149 с.

16 Michael D. Wittman. "Solving" the Blotto Game: A Computational Approach – 2011.

17 Arad, Ayala and Ariel Rubinstein. "Colonel Blotto's Top Secret Files: Multidimensional Iterative Reasoning in Action." Working Paper. 2010.

18 Rafael Hortala-Vallve, Aniol Llorente-Saguer. Pure strategy Nash equilibria in non-zero sum colonel Blotto games. 2011.

19 Новиков Д.А., Чхартишвили А.Г. Рефлексивные игры. Рефлексия и управление. 2013.

# ДОДАТОК А

Нижче буде наведено програмну реалізацію методу, що був запропонований для пошуку множини оптимальних стратегій ГПБ, описаного у розділі 4. Була використана мова програмування C#.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

class Program {
 static void Main(string[] args) {
  Console.BackgroundColor = ConsoleColor.White;
  Console.Clear();
  Console.ForegroundColor = ConsoleColor.Black;

  //Partitions.CalcAmountOfPartitions();
  SimulationOfColonelBlottoGame();

  Console.ReadKey();
 }

 public static void SimulationOfColonelBlottoGame() {
  int AmountOfResourcesOfFirPlayer = 3;
  int AmountOfResourcesOfSecPlayer = 3;
  int AmountOfObjects = 3;
  var UtilityVectorOfFirPlayer = new List < int > () {7,4,2};
  var UtilityVectorOfSecPlayer = new List < int > () {2,5,5};
```

```
        var averageAlloc = AmountOfResourcesOfSecPlayer / AmountOfObjects;
        var EnemyAllocationSuggestion = new List < double > ();
        for (var i = 0; i < AmountOfObjects; i++) EnemyAllocationSuggestion.Add(averageAlloc);
        var listOfBestAllocationOfFirPlayer = new List < List < double >> ();
        var listOfBestAllocationOfSecPlayer = new List < List < double >> ();
        var (PreferenceRatiosForFirPl, PreferenceRatiosForSecPl) = GetListOfPreferenceRatios(UtilityVectorOfFirPlayer, UtilityVectorOfSecPlayer);

        List < List < double >> tempBestAllocationsOfFirPl = new List < List < double >> ();
        var tempBestAllocationsOfSecPl = new List < List < double >> (new List < List < double >> () {
         EnemyAllocationSuggestion
        });

AddListToListOfListsIfItWasNotAddedYet(listOfBestAllocationOfSecPlayer, tempBestAllocationsOfSecPl[0]);
        double tempBestPriceOfFirPl, tempBestPriceOfSecPl;

        for (int i = 0; i < AmountOfObjects; i++) {
         Console.WriteLine($ "Iteration #{i}: ");

         foreach(var tempBestAllocOfSecPl in tempBestAllocationsOfSecPl) {
          Console.Write($ "Allocation of second player: ");
          PrintListOfItemsWithSeparator(tempBestAllocOfSecPl);
          (tempBestPriceOfFirPl, tempBestAllocationsOfFirPl) = KindOfBackpackProblem.MainFunction(
           UtilityVector: UtilityVectorOfFirPlayer,
           EnemyAllocation: tempBestAllocOfSecPl,
```

```
            AmountOfResources: AmountOfResourcesOfFirPlayer,
            AmountOfObjects: AmountOfObjects,
            PreferenceRatios: PreferenceRatiosForFirPl);
           foreach(var el in tempBestAllocationsOfFirPl) {

AddListToListOfListsIfItWasNotAddedYet(listOfBestAllocationOfFirPlayer, el);
            Console.Write($ "Best allocation of first player: ");
            PrintListOfItemsWithSeparator(el);
           }
          }

          Console.WriteLine();

          foreach(var tempBestAllocOfFirPl in tempBestAllocationsOfFirPl) {
           Console.Write($ "Allocation of first player: ");
           PrintListOfItemsWithSeparator(tempBestAllocOfFirPl);
           (tempBestPriceOfSecPl, tempBestAllocationsOfSecPl) =
KindOfBackpackProblem.MainFunction(
            UtilityVector: UtilityVectorOfSecPlayer,
            EnemyAllocation: tempBestAllocOfFirPl,
            AmountOfResources: AmountOfResourcesOfSecPlayer,
            AmountOfObjects: AmountOfObjects,
            PreferenceRatios: PreferenceRatiosForSecPl);
           foreach(var el in tempBestAllocationsOfSecPl) {

AddListToListOfListsIfItWasNotAddedYet(listOfBestAllocationOfSecPlayer, el);
            Console.Write($ "Best allocation of second player: ");
            PrintListOfItemsWithSeparator(el);
           }
           //Console.WriteLine();
```

```
        }
        Console.WriteLine();
       }
       Console.WriteLine("-----------------------------------------");
       BuildTwoPayoffMatrixForPlayers(listOfBestAllocationOfFirPlayer,
listOfBestAllocationOfSecPlayer, UtilityVectorOfFirPlayer,
UtilityVectorOfSecPlayer);
       }

       /// <summary>
       /// Makes check of all number from newList with coresponding numbers
from all list's objects.
       /// </summary>
       /// <param name="listOfLists">listOfLists</param>
       /// <param name="newList">newList</param>
       public static void AddListToListOfListsIfItWasNotAddedYet(List < List <
double >> listOfLists, List < double > newList) {
        bool existed = false;
        foreach(var el in listOfLists) {
         int i = 0, len = Math.Min(el.Count, newList.Count);
         for (; i < len; i++) {
          if (el[i] != newList[i])
           break;
         }
         if (i >= len) {
          existed = true;
          break;
         }
        }
        if (existed == false) listOfLists.Add(newList);
```

```
        }

        public static void BuildTwoPayoffMatrixForPlayers(List < List < double >> allocationsOfFirPl, List < List < double >> allocationsOfSecPl, List < int > utilityVectorOfFirPlayer, List < int > utilityVectorOfSecPlayer) {
            Console.WriteLine("Payoff matrix for first player:");
            var firListOfPairs = BuildPayoffMatrixForOnePlayer(allocationsOfFirPl, allocationsOfSecPl, utilityVectorOfFirPlayer, true);
            Console.WriteLine();
            Console.WriteLine("Payoff matrix for second player:");
            var secListOfPairs = BuildPayoffMatrixForOnePlayer(allocationsOfSecPl, allocationsOfFirPl, utilityVectorOfSecPlayer, false);
            var intersectPairsWithPayoffs = new List < Tuple < Tuple < int,
             int > , Tuple < double, double >>> ();
            for (int i = 0; i < firListOfPairs.Count; i++) {
             for (int j = 0; j < secListOfPairs.Count; j++) {
              if (firListOfPairs[i].Item1.Item1 == secListOfPairs[j].Item1.Item1 && firListOfPairs[i].Item1.Item2 == secListOfPairs[j].Item1.Item2) {
               intersectPairsWithPayoffs.Add(new Tuple < Tuple < int, int > , Tuple < double, double >> (
                firListOfPairs[i].Item1,
                new Tuple < double, double > (firListOfPairs[i].Item2, secListOfPairs[j].Item2)));
               break;
              }
             }
            }
            int limitOfStrToShow = 20;
            if (intersectPairsWithPayoffs.Count > 0) {
             Console.WriteLine();
```

```
        Console.WriteLine("Among best strategies and allocations were found next ones:");
        Console.WriteLine();
        var bestBestStrFir = new List < List < double >> ();
        var bestBestStrSec = new List < List < double >> ();
        foreach(var bestPropositions in intersectPairsWithPayoffs) {
         Console.Write("For first player: ");
         var tempAlloc = allocationsOfFirPl[bestPropositions.Item1.Item1];
         bestBestStrFir.Add(tempAlloc);
         PrintListOfItemsWithSeparator(tempAlloc);
         Console.WriteLine($ "Payoff will be equal to {bestPropositions.Item2.Item1}.");

         Console.Write("For second player: ");
         tempAlloc = allocationsOfSecPl[bestPropositions.Item1.Item2];
         bestBestStrSec.Add(tempAlloc);
         PrintListOfItemsWithSeparator(tempAlloc);
         Console.WriteLine($ "Payoff will be equal to {bestPropositions.Item2.Item2}.");
         Console.WriteLine();
        }


        var statisticList = new List < Tuple < int,
         statisticOfPermutation >> ();
        Console.WriteLine("Statistics about chosen best strategies of first player comparing with chosen best strategies of second player:");
        int ind = 0;
        foreach(var firPerm in bestBestStrFir) {
         var oneTempStat = new statisticOfPermutation();
```

```
            foreach(var secPerm in allocationsOfSecPl) {
             int tempMarker = 0;
             for (int i = 0; i < firPerm.Count; i++) {
              if (firPerm[i] > secPerm[i]) tempMarker++;
              else if (firPerm[i] < secPerm[i]) tempMarker--;
             }
             if (tempMarker == 0) oneTempStat.Draw++;
             else if (tempMarker > 0) oneTempStat.Win++;
             else oneTempStat.Lose++;
            }
            statisticList.Add(new Tuple < int, statisticOfPermutation > (ind++,
oneTempStat));
           }
           var orderedListOfFirPlStrategies = new List < double > ();
           var result = statisticList.OrderByDescending(u =>
u.Item2.Win).ThenByDescending(u => u.Item2.Draw).ThenBy(u =>
u.Item2.Lose);
           foreach(var item in result) {
            if (limitOfStrToShow-- < 0) break;
            for (int i = 0; i < bestBestStrFir[item.Item1].Count; i++)
             Console.Write(bestBestStrFir[item.Item1][i] + " ");
            Console.Write("    ");
            item.Item2.WriteItInConsole();
           }
          } else {
           var statisticList = new List < Tuple < int,
            statisticOfPermutation >> ();
           Console.WriteLine("Statistics about chosen best strategies of first player
comparing with chosen best strategies of second player:");
           int ind = 0;
```

```
            foreach(var firPerm in allocationsOfFirPl) {
             var oneTempStat = new statisticOfPermutation();
             foreach(var secPerm in allocationsOfSecPl) {
              int tempMarker = 0;
              for (int i = 0; i < firPerm.Count; i++) {
               if (firPerm[i] > secPerm[i]) tempMarker++;
               else if (firPerm[i] < secPerm[i]) tempMarker--;
              }
              if (tempMarker == 0) oneTempStat.Draw++;
              else if (tempMarker > 0) oneTempStat.Win++;
              else oneTempStat.Lose++;
             }
             statisticList.Add(new Tuple < int, statisticOfPermutation > (ind++, oneTempStat));
            }
            var orderedListOfFirPlStrategies = new List < double > ();
            var result = statisticList.OrderByDescending(u => u.Item2.Win).ThenByDescending(u => u.Item2.Draw).ThenBy(u => u.Item2.Lose);
            foreach(var item in result) {
             if (limitOfStrToShow-- < 0) break;
             for (int i = 0; i < allocationsOfFirPl[item.Item1].Count; i++)
              Console.Write(allocationsOfFirPl[item.Item1][i] + " ");
             Console.Write("    ");
             item.Item2.WriteItInConsole();
            }
           }
          }
```

```
public static List < Tuple < Tuple < int, int > , double >> BuildPayoffMatrixForOnePlayer(List < List < double >> allocationsOfFirPl, List < List < double >> allocationsOfSecPl, List < int > utilityVectorOfFirPlayer, bool isFirstPlayerReallyFirst) {
        Console.WriteLine($ "Side (strategies of {GetEngWordEqualToFirstOrSecPlayer(isFirstPlayerReallyFirst)} player):");
        PrintListOfList(allocationsOfFirPl, true);
        Console.WriteLine($ "Top (strategies of {GetEngWordEqualToFirstOrSecPlayer(!isFirstPlayerReallyFirst)} player):");
        PrintListOfList(allocationsOfSecPl, true);
        var listWithMaximums = new List < KeyValuePair < double,
         List < int >>> ();
        var matrix = new List < List < double >> ();
        var indexRow = 0;
        foreach(var firAlloc in allocationsOfFirPl) {
         var oneLineOfMatrix = new List < double > ();
         var indexColumn = 0;
         foreach(var secAlloc in allocationsOfSecPl) {
          double tempPayoff = 0;
          for (int i = 0; i < firAlloc.Count; i++) {
           if (firAlloc[i] > secAlloc[i])
            tempPayoff += utilityVectorOfFirPlayer[i];
           else if (firAlloc[i] == secAlloc[i])
            tempPayoff += utilityVectorOfFirPlayer[i] / 2.0;
          }

          if (listWithMaximums.Count <= indexColumn) {
           listWithMaximums.Add(new KeyValuePair < double, List < int >> (tempPayoff, new List < int > () {
            isFirstPlayerReallyFirst ? indexColumn : indexRow
```

```
            }));
           } else if (listWithMaximums[indexColumn].Key < tempPayoff) {
            listWithMaximums[indexColumn] = new KeyValuePair < double, List <
int >> (tempPayoff, new List < int > () {
             isFirstPlayerReallyFirst ? indexColumn : indexRow
            });
           } else if (listWithMaximums[indexColumn].Key == tempPayoff) {
            listWithMaximums[indexColumn].Value.Add(isFirstPlayerReallyFirst   ?
indexRow : indexColumn);
           }
           indexColumn++;
           oneLineOfMatrix.Add(tempPayoff);
           Console.Write($ "{tempPayoff,4}");
          }
          indexRow++;
          matrix.Add(oneLineOfMatrix);
          Console.WriteLine();
         }

         var listOfPairs = new List < Tuple < Tuple < int,
          int > , double >> ();
         for (int i = 0; i < listWithMaximums.Count; i++) {
          for (int j = 0; j < listWithMaximums[i].Value.Count; j++) {
           if (isFirstPlayerReallyFirst)
            listOfPairs.Add(new Tuple < Tuple < int, int > , double > (new Tuple <
int, int > (listWithMaximums[i].Value[j], i), listWithMaximums[i].Key));
           else
            listOfPairs.Add(new Tuple < Tuple < int, int > , double > (new Tuple <
int, int > (i, listWithMaximums[i].Value[j]), listWithMaximums[i].Key));
          }
```

```
        }
        return listOfPairs;
        }

        public static void PrintListOfItemsWithSeparator(List < double > list, string
separator = " ") {
        for (int i = 0; i < list.Count; i++)
         Console.Write(list[i] + separator);
        Console.WriteLine();
        }

        public static void PrintListOfList(List < List < double >> listOfLists, bool
withNumOnSide = false, string separator = " ") {
        int num = 1;
        foreach(var list in listOfLists) {
         if (withNumOnSide)
          Console.Write(num++ + ": ");
         PrintListOfItemsWithSeparator(list);
        }
        Console.WriteLine();
        }

        public static List < double > CalcProportion(List < double > vector) {
        var proportions = new List < double > ();
        double sum = 0;
        foreach(var el in vector) sum += el;
        foreach(var el in vector) proportions.Add(el / sum);
        return proportions;
        }
```

```
        public static(List < double > , List < double > )
GetListOfPreferenceRatios(List < int > utilityVectorOfFirPlayer, List < int >
utilityVectorOfSecPlayer) {
        var proportionsOfFirPl = CalcProportion(utilityVectorOfFirPlayer.Select <
int, double > (i => i).ToList());
        var proportionsOfSecPl = CalcProportion(utilityVectorOfSecPlayer.Select
< int, double > (i => i).ToList());

        var preLastProportionPartForFirPl = new List < double > ();
        var preLastProportionPartForSecPl = new List < double > ();

        for (int i = 0; i < utilityVectorOfFirPlayer.Count; i++) {
         preLastProportionPartForFirPl.Add(proportionsOfFirPl[i]      *      (1      -
proportionsOfSecPl[i]));
         preLastProportionPartForSecPl.Add(proportionsOfSecPl[i]      *      (1      -
proportionsOfFirPl[i]));
        }
        return                          (CalcProportion(preLastProportionPartForFirPl),
CalcProportion(preLastProportionPartForSecPl));
        }

        public static string GetEngWordEqualToFirstOrSecPlayer(bool isFirst =
true) => isFirst ? "first" : "second";
        }

        /// <summary>
        /// Class allows to make statistic about permutation.
        /// </summary>
        class statisticOfPermutation {
         public int Win { set; get; }
```

```
        public int Draw { set;  get; }
        public int Lose { set;  get; }

        public void WriteItInConsole() {
         Console.WriteLine($ "Victories - {this.Win}; Draws - {Draw}; Losses - {Lose}");
        }
       }

       /// <summary>
       /// Class has method that returns amount of partitions of number 'n' dealing on not more than 'k' parts.
       /// </summary>
       class Partitions {
        public static void CalcAmountOfPartitions(int num = 10) {
         Console.WriteLine($ "Amount of partitions for '{num}' is {amountOfPartitions(num, num)}");
         Console.WriteLine();
        }

        /// <summary>
        /// Returns amount of patitions that able to have from number 'n'.
        /// </summary>
        /// <param name="k">Deal number 'n' on not more than 'k' parts.</param>
        /// <returns>Amount of partitions of number 'n'</returns>
        public static int amountOfPartitions(int n, int k) {
         if (k > 0 && k <= n)
          return amountOfPartitions(n, k - 1) + amountOfPartitions(n - k, k);
         else if (k > n) {
          return amountOfPartitions(n, n);
```

```
  } else if (k == 0 && n == 0) {
   return 1;
  } else return 0;
 }
}

/// <summary>
/// Class has method that searches for solution of kind of backpack problem.
/// </summary>
class KindOfBackpackProblem {
 private static List < int > bestAllocation;
 private static List < double > bestRealAllocation;
 private static List < List < double >> bestRealAllocationsList;
 private static double bestPrice;
 private static List < int > inequalitySignsForBestAllocation;
 private static double epsilon;

 private static List < int > _UtilityVector { get; set; }
 private static List < double > _EnemyAllocation;
 private static int _AmountOfResources;
 private static int _AmountOfObjects;

 private static List < double > _PreferenceRatios;

 /// <summary>
 /// MainFunction solves task with kind of backpack problem.
 /// </summary>
 /// <param name="UtilityVector">vector with payoffs of player for wins at
object</param>
```

```
        /// <param name="EnemyAllocation">Vector with allocation of enemy's resources</param>
        /// <param name="AmountOfResources">Amount of player's resources</param>
        /// <param name="AmountOfObjects">Amount of objects in game</param>
        /// <param name="PreferenceRatios">Preference ratios of objects for player</param>
        /// <returns>Turple with best payoff for player and strategy that allows to get it.</returns>
        public static(double, List < List < double >> ) MainFunction(
         List < int > UtilityVector,
         List < double > EnemyAllocation,
         int AmountOfResources,
         int AmountOfObjects,
         List < double > PreferenceRatios) {
         _UtilityVector = UtilityVector;
         _EnemyAllocation = EnemyAllocation;
         _AmountOfResources = AmountOfResources;
         _AmountOfObjects = AmountOfObjects;
         _PreferenceRatios = PreferenceRatios;

         if ((bestAllocation != null) && bestAllocation.Count > 0)
          bestAllocation.Clear();
         if ((bestRealAllocation != null) && bestRealAllocation.Count > 0)
          bestRealAllocation.Clear();
         if ((bestRealAllocationsList != null) && bestRealAllocationsList.Count > 0)
          bestRealAllocationsList.Clear();
```

```
        if ((inequalitySignsForBestAllocation != null) &&
inequalitySignsForBestAllocation.Count > 0)
         inequalitySignsForBestAllocation.Clear();
        bestPrice = 0;
        epsilon = 0;

        var (price, allocations) = MaximizationOfBenefit(
         UtilityVector: UtilityVector,
         EnemyAllocation: EnemyAllocation,
         AmountOfResources: AmountOfResources,
         AmountOfObjects: AmountOfObjects);

        var appropriateAllocations = new List < List < double >> ();
        foreach(var el in allocations)
        appropriateAllocations.Add(new List < double > (el));
        return (price, appropriateAllocations);
       }

       public static(double, List < List < double >> ) MaximizationOfBenefit(List
< int > UtilityVector, List < double > EnemyAllocation, int AmountOfResources,
int AmountOfObjects) {
        var flagsForUtilityFunc = GetAllNSets(AmountOfObjects);
        flagsForUtilityFunc.Reverse();
        var allInequalitySignCombs = GetAllNSets(AmountOfObjects, 3);

        foreach(var inequalitySignComb in allInequalitySignCombs) {
         // check  reality of inequalitySigns
         int posibleUnused = 0, usedInAlloc = 0;
         bool flagOfSignMore = false;
         int i;
```

```
            for (i = 0; i < AmountOfObjects && usedInAlloc <= AmountOfResources; i++) {
             switch (inequalitySignComb[i]) {
              case 0:
               posibleUnused += (int) EnemyAllocation[i] - 1;
               break;
              case 1:
               usedInAlloc += (int) EnemyAllocation[i];
               break;
              case 2:
               usedInAlloc += (int) EnemyAllocation[i] + 1;
               flagOfSignMore = true;
               break;
             }
            }
            if (usedInAlloc > AmountOfResources || (!flagOfSignMore && posibleUnused < AmountOfResources))
             continue;

            foreach(var oneFlagForUtilityFunc in flagsForUtilityFunc) {
             CheckSet(oneFlagForUtilityFunc, inequalitySignComb);
             CheckSet(oneFlagForUtilityFunc, inequalitySignComb, true);
            }
           }

           //ShowInequalitySignsForBestAllocation();
           return (bestPrice, bestRealAllocationsList);
          }
```

```
        private static(double, List < double > ) CalcWeigth(List < int > vector, List
< int > inequalitySigns) {
         double sumW = 0,
          benefitCounter = 0;
         var epsList = new List < double > ();
         for (var i = 0; i < vector.Count; i++) {
          if (inequalitySigns[i] >= 1)
           sumW += vector[i] * _EnemyAllocation[i];
          if (inequalitySigns[i] > 1)
           benefitCounter++;
          epsList.Add(0);
         }
         epsilon = _AmountOfResources - sumW;
         if (epsilon > 0) {
          var partEps = epsilon / benefitCounter;
          for (var i = 0; i < vector.Count; i++) {
           if (inequalitySigns[i] > 1)
            epsList[i] = partEps;
          }
         }
         return (sumW, epsList);
        }

        private static List < double > FindAppropriateAllocation(List < int > vector,
List < int > inequalitySigns, bool equalResourceAllocation) {
         List < double > allocation = new List < double > ();
         double sumW = 0;
         int benefitObjects = 0;
         var dict = new Dictionary < int,
          double > ();
```

```
for (var i = 0; i < vector.Count; i++) {
 double temp = 0;
 if (inequalitySigns[i] >= 1) {
  temp = vector[i] * _EnemyAllocation[i];
  sumW += temp;
  if (inequalitySigns[i] > 1) {
   benefitObjects++;
   dict.Add(i, _PreferenceRatios[i]);
  }
 }
 allocation.Add(temp);
}
dict.OrderBy(i => i.Value);
epsilon = _AmountOfResources - sumW;
while (epsilon > 0 && dict.Count > 0) {
 if (equalResourceAllocation) {
  int part = (int) epsilon / dict.Keys.Count;
  foreach(var el in dict.Keys) {
   allocation[el] += part;
   epsilon -= part;
  }
 } else {
  var tempEps = epsilon;
  foreach(var el in dict.Keys) {
   int part = (int)(tempEps * dict[el]);
   allocation[el] += part;
   epsilon -= part;
  }
 }
 foreach(var el in dict.Keys) {
```

```
  if (epsilon > 0) {
   allocation[el]++;
   epsilon--;
  } else break;
 }
}
if (epsilon > 0) {
 var partEpsilon = epsilon / benefitObjects;
 var benefitCounter = 0;
 for (var i = 0; i < vector.Count && partEpsilon > 0; i++) {
  if (inequalitySigns[i] == 2) {
   allocation[i] += partEpsilon;
   benefitCounter++;
  }
 }
 if (benefitCounter != benefitObjects)
  allocation.Clear();
}
return allocation;
}

private static double CalcBenefitOfAllocation(List < double > allocation) {
 double sumW = 0;
 for (var i = 0; i < allocation.Count; i++) {
  if (allocation[i] == _EnemyAllocation[i] && _EnemyAllocation[i] != 0)
   sumW += _UtilityVector[i] / 2.0;
  if (allocation[i] > _EnemyAllocation[i])
   sumW += _UtilityVector[i];
 }
 return sumW;
```

```
        }

        private static void CheckSet(List < int > items, List < int > inequalitySigns,
bool equalResourceAllocation = false) {
         if (bestAllocation == null) {
          var (cW, eps) = CalcWeigth(items, inequalitySigns);
          if (cW <= _AmountOfResources) {
           var alloc = FindAppropriateAllocation(items, inequalitySigns,
equalResourceAllocation);
           bestPrice = CalcBenefitOfAllocation(alloc);
           bestAllocation = items;
           bestRealAllocation = alloc;
           inequalitySignsForBestAllocation = inequalitySigns;
          }
         } else {
          var (cW, epsList) = CalcWeigth(items, inequalitySigns);
          var cC = CheckConditions(items, inequalitySigns, epsList);
          if (cW <= _AmountOfResources && cC) {
           var alloc = FindAppropriateAllocation(items, inequalitySigns,
equalResourceAllocation);
           if (alloc.Count == 0) return;
           var cB = CalcBenefitOfAllocation(alloc);
           if (cB > bestPrice) //more than one strategy may have best price ( == )
           {
            bestAllocation = items;
            bestPrice = cB;
            inequalitySignsForBestAllocation = inequalitySigns;
            bestRealAllocation = alloc;

            bestRealAllocationsList.Clear();
```

```
Program.AddListToListOfListsIfItWasNotAddedYet(bestRealAllocationsList,
alloc);
            }
            if (cB == bestPrice) {
             bestAllocation = items;
             bestPrice = cB;
             inequalitySignsForBestAllocation = inequalitySigns;
             bestRealAllocation = alloc;
             if (bestRealAllocationsList == null)
              bestRealAllocationsList = new List < List < double >> ();

Program.AddListToListOfListsIfItWasNotAddedYet(bestRealAllocationsList,
alloc);
             }
            }
           }
          }

        public static bool CheckConditions(List < int > allocation, List < int >
inequalitySigns, List < double > epsList) {
           for (int i = 0; i < _AmountOfObjects; i++) {
            switch (inequalitySigns[i]) {
             case 0: // less
              if (allocation[i] * _EnemyAllocation[i] + epsList[i] >=
_EnemyAllocation[i])
               return false;
              break;
             case 1: // equal
```

```
            if (allocation[i] * _EnemyAllocation[i] + epsList[i] ==
_EnemyAllocation[i])
             break;
            else return false;
           case 2: // more
            if (allocation[i] * _EnemyAllocation[i] + epsList[i] <=
_EnemyAllocation[i])
             return false;
            break;
          }
         }
         var sum = 0;
         foreach(var i in allocation) {
          if (i > 1) return false;
          sum += i;
         }
         if (sum > _AmountOfResources)
          return false;
         return true;
        }

        public static void ShowInequalitySignsForBestAllocation() {
         Console.Write("InequalitySignsForBestAllocation: ");
         ShowInequalitySigns(inequalitySignsForBestAllocation);
        }

        public static void ShowInequalitySigns(List < int > inequalitySigns) {
         foreach(var el in inequalitySigns) {
          Console.Write($ "{ConvertNumberIntoInequalitySign(el)} ");
         }
```

```
 Console.WriteLine();
 }

 private static string ConvertNumberIntoInequalitySign(int
numberAsInequalitySign) {
 switch (numberAsInequalitySign) {
  case 0:
   return "<";
  case 1:
   return "=";
  case 2:
   return ">";
 }
 return """;
 }

 private static List < List < int >> GetAllNSets(int n, int root = 2) {
 var allNSets = new List < List < int >> ();
 var allowedSet = new List < int > ();
 for (int i = 0; i < root; i++) {
  allowedSet.Add(i);
 }
 for (int i = 0; i < n; i++) {
  if (i != 0) {
   var tempListOfLists = new List < List < int >> ();
   foreach(var el in allNSets) {
    var newList = new List < int > (el);
    for (int j = 0; j < root; j++) {
     var innerList = new List < int > (newList);
     innerList.Add(allowedSet[j]);
```

```
      tempListOfLists.Add(innerList);
     }
    }
    allNSets.Clear();
    allNSets.AddRange(tempListOfLists);
   } else {
    foreach(var el in allowedSet) {
     var newList = new List < int > () {
      el
     };
     allNSets.Add(newList);
    }
   }
  }
  return allNSets;
 }
}
```